IBM **Technical Newsletter**

File No. GENL-25

Re: Form No. C24-3322-0

This Newsletter No. N24-0300

Date: March 1, 1965

Previous Newsletter Nos. N24-0280

Replacement pages for Fortran IV Language Specifications, Program Specifications, and Operating Procedures, IBM 1401, 1440, and 1460, Form C24-3322-0.

To bring your publication up to date, please replace the following pages with the corresponding pages attached to this Newsletter. Changes are indicated by a vertical line at the left of the affected text and by a dot (●) at the left of the figure title of the affected figure.

Pages	Pages Added	Pages Deleted
5 and 6	44.1 and 44.2	84 and 85
19 and 20	44.3 and 44.4	
21 and 22		
25 and 26		
33 and 34		
39 and 40		
45 and 46		
51 and 52		
53 and 54		
55 and 56		
69 and 70		
75 and 76		
77 and 78		
79 and 80		
81 and 82		
83 and 86		

Please insert this page to indicate that your publication now includes the modified pages issued with Technical Newsletter:

Form	Pages	Date
N24-0280	1, 17, 43, 44, 50, 57, 64, 65, 88, 89, 90	11/30/64
N24-0300	5, 6, 20, 21, 22, 25, 33, 39, 44.1, 44.2, 44.3, 44.4, 46, 51, 53, 55, 69, 75, 76, 78, 79, 81, 82, 83, 86	3/1/65

IBM Corp., Product Publications Dept., Endicott, N. Y. 13764

PRINTED IN U.S.A.

This publication contains the language specifications, program specifications, and operating procedures for the Fortran IV programming system for IBM 1401, 1440, and 1460. In this publication, the term *Fortran system* refers to *1401/1440/1460 Fortran IV,* program numbers 1401-FO-051 (Disk Resident System) or 1401-FO-052 (Tape Resident System).

This publication is divided into three major sections, *language specifications, program specifications,* and *operating procedures.*

The language specifications section describes the coding of a Fortran program. The content of this section is presented with the assumption that the programmer is familiar with the information in the *Fortran General Information Manual,* Form F28-8074.

The program specifications section describes the Fortran system. Included in the section are such topics as a description of the *System Control Program* (the controlling element of the Fortran system), a description of the *Fortran Processor Program,* and a detailed description of the results of system operations. Although this section is directed primarily to the programmer, the machine operator should review the section for an understanding of the system.

The third section, operating procedures, contains such topics as preparing processor jobs, changing file assignments for processor jobs, and running processor jobs. The last part of the section outlines the procedures to follow in building a Fortran system. For the convenience of both programmer and machine operator, all control cards are summarized in *Appendix I.*

While the third section is directed primarily to the machine operator, it is recommended that the programmer review the content of the complete section. The programmer should particularly note the parts of the section dealing with preparing processor jobs and changing file assignments.

Related Information

The following Systems Reference Library publications contain additional information relating to the use of the Fortran system. It is recommended that these publications be available to the user for reference purposes.

Fortran General Information Manual, Form F28-8074.

Disk Utility Programs Specifications for IBM 1401, 1440, and 1460 (with 1301 and 1311), Form C24-1484.

Disk Utility Programs Operating Procedures for IBM 1401 and 1460 (with 1301 and 1311), Form C24-3105, or *Disk Utility Programs Operating Procedures for IBM 1440 (with 1301 and 1311),* Form C24-3121.

Definitions of Key Terms

In order to clarify the meaning of special terms when used in this publication, the following definitions are given. Standard terms are defined in *Glossary of Information Processing,* Form C20-8089.

Batched Files. Logical files whose contents represent one or more sequential sets of input to or output from the Fortran system.

Card Boot. A card deck, supplied as part of the Fortran system program deck, that is used to start system operations.

Compiler. The program that translates Fortran symbolic statements directly into relocatable machine language. This process is called a *compilation.*

Job. An operation or sequence of operations that are to be performed by the Fortran system.

Logical Files. Input/output devices and/or areas that are used by the Fortran system.

Object-time. A term describing those elements or processes related to the execution of a machine-language object program.

Operation. A basic unit of work to be performed by one of the components of the system.

Stack. A set of one or more jobs that is to be processed during the same machine run.

System. The set of programs made up of the elements required for compiling and/or executing user-programs.

[ ] Brackets contain an option that may be chosen. Braces contain options, one of which must be chosen.

Machine Requirements

To process a Fortran source program, the following minimum machine configurations are specified.
An IBM 1401 system with:
12,000 positions of core storage
Advanced-Programming feature

High-Low-Equal-Compare feature

One IBM 1311 Disk Storage Drive, or four IBM 729 Magnetic Tape Units, or four IBM 7330 Magnetic Tape Units, or a combination of four IBM 729 Magnetic Tape Units and IBM 7330 Magnetic Tape Units

One IBM 1402 Card Read-Punch

One IBM 1403 Printer

An IBM 1440 system with:

12,000 positions of core storage

Indexing-and-Store-Address-Register feature

One IBM 1301 Disk Storage, or one IBM 1311 Disk Storage Drive

One IBM 1442 Card Reader

One IBM 1443 Printer

An IBM 1460 system with:

12,000 positions of core storage

Indexing-and-Store-Address-Register feature

One IBM 1301 Disk Storage, or one IBM 1311 Disk Storage Drive, or four IBM 729 Magnetic Tape Units, or four IBM 7330 Magnetic Tape Units, or a combination of four IBM 729 Magnetic Tape Units and IBM 7330 Magnetic Tape Units

One IBM 1402 Card Read-Punch

One IBM 1403 Printer

To load and execute object programs generated by the Fortran system, the following minimum machine requirements are specified.

An IBM 1401 system with:

12,000 positions of core storage (or more if required by the object program)

Advanced-Programming feature

High-Low-Equal-Compare feature

One IBM 1311 Disk Storage Drive, or one IBM 729 Magnetic Tape Unit, or one IBM 7330 Magnetic Tape Unit (for residence of the Fortran system, including the library of the relocatable subprograms)

One IBM 1402 Card Read-Punch

One IBM 1403 Printer

Sense switches, if required by the object program

A program loading device, which could be an IBM 1402 Card Read-Punch (the same as previously required), an IBM 1311 Disk Storage Drive (may be the same as that required for system residence), an IBM 729 Magnetic Tape Unit (must *not* be the same unit as that required for system residence), or an IBM 7330 Magnetic Tape Unit (must *not* be the same unit as that required for system residence)

Additional input/output devices as required by the object program

An IBM 1440 system with:

12,000 positions of core storage (or more if required by the object program)

Indexing-and-Store-Address-Register feature

One IBM 1301 Disk Storage, or one IBM 1311 Disk Storage Drive (for residence of the Fortran system, including the library of the relocatable subprograms)

One IBM 1442 Card Reader

One IBM 1443 Printer

Sense switches, if required by the object program

A program loading device, which could be an IBM 1442 Card Reader (the same as previously required), an IBM 1301 Disk Storage or an IBM 1311 Disk Storage Drive (may be the same as that required for system residence)

Additional input/output devices as required by the object program

An IBM 1460 system with:

12,000 positions of core storage (or more if required by the object program)

Indexing-and-Store-Address-Register feature

One IBM 1301 Disk Storage, or one IBM 1311 Disk Storage Drive, or one IBM 729 Magnetic Tape Unit, or one IBM 7330 Magnetic Tape Unit (for residence of the Fortran system, including the library of the relocatable subprograms)

One IBM 1402 Card Read-Punch

One IBM 1403 Printer

Sense switches, if required by the object program

A program loading device, which could be an IBM 1402 Card Read-Punch (the same as previously required), or an IBM 1301 Disk Storage or an IBM 1311 Disk Storage Drive (may be the same as that required for system residence), or an IBM 729 Magnetic Tape Unit (must *not* be the same unit as that required for system residence), or an IBM 7330 Magnetic Tape Unit (must *not* be the same unit as that required for system residence)

Additional input/output devices as required by the object program

The Fortran system can use the following devices, if available.

IBM 1444 Card Punch

IBM 1447 Console without a buffer feature

would transfer the first record with the specification I2,3E12.4, the second record with the specification 2F10.3,3F9.4, and all remaining records with the specification 10F12.4. The repetition starts at the last left parenthesis, including a repeater, if present.

If data items remain to be transferred after the format specification has been completely interpreted, the specification repeats after the last left parenthesis. Group repetition applies again if it is present. Consider this example.

```
FORMAT(3E10.3,2(I2,2F12.4,D28.17))
```

If more items are to be transferred after this format specification has been completely used, the specification repeats with I2 after the last left parenthesis. The 2 preceding the parenthesis, indicating group repetition, applies again.

Carriage Control

Carriage control characters must appear in the first position of the output record if the record is to be printed. The valid characters used to achieve the desired results follow.

Character	*Result*
blank	No space before printing, that is, single space printing
0	One space before printing, that is, double space printing
1-9	Skip to channel 1-9 before printing as indicated

The printer carriage will space to the next line immediately following each print operation. The control character does not appear in the printed record.

The carriage control character is also in output tape or disk records that are to be used for off-line tape- or disk-to-printer operations.

The carriage control character can be provided by a 1H*x* as the first field specification of a FORMAT statement, where *x* is the carriage control character. For example, the field specification 1H6 causes a 6 to be inserted in the high-order position of the output record. This in turn causes the printer carriage to skip to channel 6 before printing.

When alphameric text is specified for the high-order field of an output record, the carriage control character can be included in the alphameric field specification. For example, if X SQUARED is to occupy the high-order position of an output record, the specification could be 11H6XbSQUAREDb. The specification 11H6 XbSQUAREDb causes the printer carriage to skip to channel 6 before printing the record. This specification could also be written 1H6,10HXbSQUAREDb.

FORMAT Statements Read In at Object Time

Fortran can accept a variable FORMAT address. This permits the specification of a FORMAT for an I/O list at object time. In order that this be accomplished, two factors must be taken into account.

First, the name of the variable FORMAT specification must appear in a DIMENSION statement, even if the array size is only 1.

Second, the format read in at object time under A-conversion must take the same form as a source program FORMAT statement, except that the word FORMAT is omitted. The variable format begins with a left parenthesis and ends with a right parenthesis.

Consider this example. A, B, and array C are converted and stored according to the FORMAT specifications that are read into the array FMT at object time.

```
      DIMENSION FMT (12)
    1 FORMAT (8A10)
      READ (1, 1) (FMT (I), I = 1, 8)
      READ (1, FMT) A, B, (C (I), I = 1, 5)
```

Assume that the first data card containing the FORMAT statement is of the form

```
(b2F10.8/b(E10.2))bb . . .
```

This implies that (b2F10.8/b will be stored in FMT(1) and (E10.2))bb will be stored in FMT(2), assuming $f + 2 = 10$. The remaining characters will be stored in FMT(3) . . . . This further implies that A and B will be read according to the F10.8 FORMAT specification and C(1), . . . , C(5) will be read according to the E10.2 FORMAT specification.

Edited Input Data

Edited input data to an object program is contained in records that conform to the following specifications:

1. The data must correspond in order, type, and field width to the field specifications in the FORMAT statement. Reading of data starts with the first character position.
2. Plus signs are indicated by a blank or a preceding + (12-zone punch). Minus signs are indicated by a preceding — (11-zone punch). Signs are not indicated over the units position of the field.
3. Blanks in numeric fields are regarded as zeros.
4. Numbers for E- and F-conversion can contain any number of digits, but only the high-order f digits will be retained. The number is rounded to f digits of accuracy. The absolute value of the number must be between the limits 10^{-100} and $(1 - 10^{-f}) \times 10^{99}$, or be zero. Numbers for I-conversion must be right

justified (trailing blanks are regarded as zeros) and can contain any number of digits, but only the low-order *k* digits are retained.

5. Numbers for E-conversion need not have four columns devoted to the exponent field. The start of the exponent field must be marked by an E, or if the E is omitted, by a + or — (not a blank). Valid forms for the exponent field are:

   E+02, Eb02, Eb2, E+2, E2, +02, +2, E−22, E−2, −2.

6. Numbers for E- and F-conversion need not have the decimal point punched. The format specification will supply the required decimal point. For example, the number —09321+2 with the specification E12.4 will be treated as though the decimal point had been punched between the 0 and the 9. If a decimal point is punched, its position overrides the position indicated in the FORMAT specification.

7. A 7-8 punch in column one indicates an immediate return to the System Control Program.

Unedited Data

Unedited data may be stored on a disk unit or on magnetic tape. The length of each complete unedited record is defined by the number of items in the I/O list. Unedited records that are read from either a disk unit or magnetic tape must have been written by a Fortran WRITE statement. That is, the lists for the READ and WRITE statements must be of the same length. Further, the types of the list items must be in a one-to-one correspondence. For example, if the list of the WRITE statement were in the form

A, B, C

and the list of the READ statement were in the form

X, Y, Z

the types of A and X must match, the types of B and Y must match, and the types of C and Z must match.

The information is read or written using the internal representation of data values with no conversion.

The Fortran object-time routines may physically segment complete records into partial records.

General Input/Output Statements

The input/output devices available on the object machine are the only devices that can be referenced in an I/O statement. In particular, a disk unit must be available in order to use the DEFINE FILE, FIND, and the random form of the READ and WRITE statements. The sequential READ and WRITE, ENDFILE, BACKSPACE and REWIND statements can reference a disk unit area only if the area has not been referenced in a DEFINE FILE statement. Sequential operations on disk and tape are effectively the same. The tape resident Fortran system can compile statements referencing a disk unit, but cannot use a disk unit at compile time, at load time, or at execution time.

The READ Statement

The READ statement designates input. This statement is used to transfer data from input devices to the computer.

General Forms.

READ *(i, n) list*
READ *(i) list*
READ *(j ′ e, n) list*
READ *(j ′ e) list*

i is an unsigned one digit integer constant or an integer variable that specifies a specific logical file to be used for data input.

n is the statement number of a FORMAT statement or a real array name that describes the data to be transferred.

list is an input list.

j is an unsigned one digit integer constant or an integer variable that specifies a specific logical file on a disk unit whose data input is to be accessed randomly.

′ is a 4-8 punch (equivalent to the @ symbol).

e is an unsigned integer constant, integer variable, or integer expression that specifies a specific record within logical file *j*.

Examples.

```
READ (5, 10) A, B, (D (J), J = 1, 10)
READ (N, 10) K, D (J)
READ (3) (A(J), J = 1, 10)
READ (N) (A(J), J = 1, 10)
READ (6 ′ 55, 15)
READ (I ′ J, 22)
READ (I ′ J + 5) X, Y, Z
```

The READ *(i,n) list* statement is used when the logical file contains edited information and is assigned to a card reader, or to a console printer, or to a tape unit, or to a disk unit whose records are to be selected sequentially. The statement causes the edited information to be read from the logical file *i* according to FORMAT statement *n*. Successive records are read in accordance with the FORMAT statement *n* until all the data items in the I/O list have been read, converted, and stored in the location specified by the I/O list.

The READ *(i) list* statement is used when the logical file contains unedited information and is assigned to a tape unit or to a disk unit whose records are to be processed sequentially. The statement causes the unedited information to be read from the logical file *i* starting with the record at the current position of the device to

which logical file *i* is assigned. Only one record is read. The record is read completely only if the list specifies as many variables as the number of values in the record. Unedited records that are to be read in by a Fortran program should have been written by a Fortran program that used the same degrees of precision; that is, the values of *k* are the same for both programs, and the values of *f* are the same for both programs. An unedited record to be read can be divided into several parts by a Fortran I/O routine.

The READ *(j ' e,n) list* statement is used when the logical file contains edited information and is assigned to a disk unit whose records may be processed randomly. The statement causes the edited information to be read according to FORMAT statement *n* starting with record *e* within logical file *j*. Successive records are read in accordance with the FORMAT statement *n* until all the data items in the I/O list have been read, converted, and stored in the location specified by the I/O list. Each record should have no more characters than the maximum specified in the DEFINE FILE statement for the logical file *j*.

The READ *(j ' e) list* statement is used when the logical file contains unedited information and is assigned to a disk unit whose records may be processed randomly. The statement causes the unedited information to be read from record *e* within logical file *j*. Only one record is read. The record is read completely only if the list specifies as many variables as the number of values in the record. Unedited records that are to be read in by a Fortran program should have been written by a Fortran program that used the same degrees of precision; that is, the values of *k* are the same for both programs, and the values of *f* are the same for both programs. To contain all of the list data, the record to be read can be divided into several parts by a Fortran I/O routine, consistent with the description of logical file *j* in a DEFINE FILE statement. Each record should have no more data values than the maximum specified in the DEFINE FILE statement for the logical file *j*.

The WRITE Statement

The WRITE statement designates output. This statement is used to transfer data from the computer to output devices.

General Forms.

WRITE *(i,n) list*
WRITE *(i) list*
WRITE *(j ' e, n) list*
WRITE *(j ' e) list*

i is an unsigned one digit integer constant or an integer variable that specifies a specific logical file to be used for data output.

n is the statement number of a FORMAT statement or a real array name that describes the data to be transferred.

list is an output list.

j is an unsigned one digit integer constant or an integer variable that specifies a specific logical file on a disk unit that is to be accessed randomly for data output.

' is a 4-8 punch (equivalent to the @ symbol).

e is an unsigned integer constant, integer variable, or integer expression that specifies a specific record within logical file *j*.

Examples.

```
WRITE (6,  10)A, B, (C(J), J = 1, 10)
WRITE (N, 11) K, D (J)
WRITE (2) (A(J), J = 1, 10)
WRITE (M) A, B, C
WRITE (9 ' 55, 15)
WRITE (I ' J, 22)
WRITE (I ' J + 5)
```

The WRITE *(i,n) list* statement is used when the logical file is to contain edited information and is assigned to a card punch, or to a console printer, or to a tape unit, or to a disk unit on which records are to be written sequentially. The statement causes the edited information to be output on logical file *i* according to FORMAT statement *n*. Successive records are output in accordance with the FORMAT statement *n* until all the data items in the I/O list have been converted and output.

The WRITE *(i) list* statement is used when the logical file is to contain unedited information and is assigned to a tape unit or to a disk unit on which records are to be written sequentially. The statement causes the unedited information to be written on logical file *i* starting with the record at the current position of the device to which logical file *i* is assigned. Only one record is written. The record is written completely only if the list specifies as many variables as the number of values in the record. The record to be written can be divided into several parts by a Fortran I/O routine.

The WRITE *(j ' e,n) list* statement is used when the logical file is to contain edited information and is assigned to a disk unit on which records are to be written randomly. The statement causes the edited information to be written according to FORMAT statement *n* starting with record *e* within logical file *j*. Successive records are written in accordance with the FORMAT statement *n* until all the data items in the I/O list have been converted and written. Each record should have no more characters than the maximum specified in the DEFINE FILE statement for the logical file *j*.

The WRITE *(j ' e) list* statement is used when the logical file is to contain unedited information and is assigned to a disk unit on which records are to be written randomly. The statement causes the unedited information to be written starting with record *e* within logical

file *j*. Only one record is written. The record is written completely only if the list specifies as many variables as the number of values in the record. To contain all of the list data, the record to be written can be divided into several parts by a Fortran I/O routine, consistent with the description of logical file *j* in a DEFINE FILE statement. Each record should have no more data values than the maximum specified in the DEFINE FILE statement for file *j*.

Manipulative Input/Output Statements

The FIND, END FILE, REWIND, and BACKSPACE statements manipulate the logical files that are used by the object program. The END FILE, REWIND, and BACKSPACE statements must *not* be used to reference logical files that are referenced in a DEFINE FILE statement, or logical files that are assigned to a card reader, a card punch, or a printer.

The FIND Statement

The FIND statement is used to initiate the positioning of the access mechanism when the logical file is assigned to a disk unit.

General Form. FIND *(j ' e)*

j is an unsigned one digit integer constant or an integer variable that specifies a specific logical file on a disk unit that contains data input or output to be selected randomly.

' is a 4-8 punch (equivalent to the @ symbol).

e is an unsigned integer constant, integer variable, or integer expression that specifies a specific record within logical file *j*.

The purpose of the FIND statement is to enable the programmer to increase the speed at which the object program is executed. The FIND statement may substantially reduce the seek time required by the next READ or WRITE statement, provided that it references the same record *e* within logical file *j*. When the statement is used, it starts positioning the access mechanism to locate the record *e* within logical file *j* while permitting computation to proceed concurrently. The greater the separation between the FIND statement and the following READ or WRITE statement, the greater the concurrent processing time.

The END FILE Statement

General Form. END FILE *i*

i is an unsigned integer constant or an integer variable that refers to a specific logical file.

Examples.

```
END FILE 3
END FILE N
```

The END FILE *i* statement causes an end-of-file indication to be written on the logical file *i*. If the logical file is assigned to a tape unit, a tape mark is written. If the logical file is assigned to a disk unit, an end-of-file record, 1bEOF, is written. Either indication is recognized as an end-of-file condition when sensed by a READ statement, and can be tested by using the standard subprogram EOF.

The REWIND Statement

General Form. REWIND *i*

i is an unsigned integer constant or an integer variable that refers to a specific logical file.

Examples.

```
REWIND 3
REWIND N
```

The REWIND *i* statement causes logical file *i* to be initialized to its starting point. If the logical file is assigned to a tape unit, the tape will be rewound. If the logical file is assigned to a disk unit, the START address of the disk area is obtained.

The BACKSPACE Statement

General Form. BACKSPACE *i*

i is an unsigned integer constant or an integer variable that refers to a specific logical file.

Example.

```
BACKSPACE 3
```

The BACKSPACE *i* statement causes logical file *i* to "backspace" one complete record. If the logical file contains edited records, one record is a complete record. If the logical file contains unedited records, there can be more than one partial record making up a complete record.

If the logical file is assigned to a tape unit, the tape is physically backspaced. If the logical file is assigned to a disk unit, a disk-address is appropriately decreased.

Input/Output Specification Statement

The DEFINE FILE Statement

The DEFINE FILE statement is used for logical files that are assigned to disk units such that records may be accessed randomly. The tape resident Fortran system

Function	Definition	Number of Arguments	Name	Type of Argument	Type of Function
Exponential	$_e$Arg	1	EXP	Real	Real
Natural logarithm	$\log_e$(Arg)	1	ALOG	Real	Real
Common logarithm	$\log_{10}$(Arg)	1	ALOG10	Real	Real
Arctangent	arctan (Arg)	1	ATAN	Real	Real
Trigonometric sine	sin (Arg)	1	SIN	Real	Real
Trigonometric cosine	cos (Arg)	1	COS	Real	Real
Square root	(Arg) $^{1/2}$	1	SQRT	Real	Real
Absolute value	|Arg|	1	ABS IABS	Real Integer	Real Integer
Truncation	Sign of Arg applied to largest integer≤|Arg|	1	AINT INT	Real Real	Real Integer
Remaindering (see note below)	Arg_1 (mod Arg_2)	2	AMOD MOD	Real Integer	Real Integer
Choosing largest algebraic value	Max (Arg_1, Arg_2, ...)	≥2	AMAX0 AMAX1 MAX0 MAX1	Integer Real Integer Real	Real Real Integer Integer
Choosing smallest algebraic value	Min (Arg_1, Arg_2, ...)	≥2	AMIN0 AMIN1 MIN0 MIN1	Integer Real Integer Real	Real Real Integer Integer
Float	Conversion from integer to real	1	FLOAT	Integer	Real
Fix	Conversion from real to integer	1	IFIX	Real	Integer
Transfer of sign	Sign of Arg_2 applied to |Arg_1|	2	SIGN ISIGN	Real Integer	Real Integer
Positive difference	Arg_1-Min (Arg_1, Arg_2)	2	DIM IDIM	Real Integer	Real Integer

Note: The function MOD (Arg_1, Arg_2) is defined as $Arg_1 - \left[Arg_1/Arg_2\right] Arg_2$ where $\left[x\right]$ is the integral part of x.

Figure 7. Predefined Library Functions

General Form	*Function*
LINK (3H*phase-name*) *or* LINK (*r*)	*phase-name* stands for a three-character phase-name that indicates the presence of a program in the absolute format stored as a phase on the SYSTEM file of the Fortran system. The phase (program) is loaded and execution of the program begins. *r*, a real variable, has an alphameric value whose rightmost three characters are used as the phase-name.

General Form	*Function*
EOF (*m*)	*m*, a logical variable, is assigned a value of .TRUE. if the most recently executed READ statement sensed an end-of-file condition; otherwise, *m* is .FALSE.. An end-of-file condition is indicated by 1bEOF, a tape mark, or the last card. If a READ statement is issued after the last-card condition, no halt occurs; rather, blank input is interpreted.

Defining Subprograms

This section describes the FUNCTION statement that is used to name FUNCTION subprograms, and the SUBROUTINE statement that is used to name SUBROUTINE subprograms.

The FUNCTION Statement

The FUNCTION statement is used to name FUNCTION subprograms and must be the first statement of a FUNCTION subprogram. It cannot appear anywhere else in the subprogram nor can it appear in a main program.

General Forms.

FUNCTION *name* ($a_1, a_2, \ldots, a_n$)
REAL FUNCTION *name* ($a_1, a_2, \ldots, a_n$)
INTEGER FUNCTION *name* ($a_1, a_2, \ldots, a_n$)
LOGICAL FUNCTION *name* ($a_1, a_2, \ldots, a_n$)

name is the symbolic name of the single-valued function subprogram.

$a_1, a_2, \ldots, a_n$ are the dummy arguments of the function and can be nonsubscripted variable names, or array names, or the dummy names of SUBROUTINE, or other FUNCTION subprograms. There must be at least one argument in a FUNCTION subprogram.

The type of function can be explicitly stated by the inclusion of the word REAL, INTEGER, or LOGICAL before the word FUNCTION, as shown in the preceding formats.

Examples.

```
FUNCTION ARCSIN (RADIAN)
REAL FUNCTION ROOT (A, B, D)
INTEGER FUNCTION CONST (ING, SG)
LOGICAL FUNCTION IFTRU (D, E, F)
```

The FUNCTION subprogram can contain any Fortran statement except a SUBROUTINE statement or another FUNCTION statement.

The name of the function must appear at least once as a variable on the left side of an arithmetic statement, or as an element of an input list.

Consider the following example.

```
FUNCTION CALC (A, B)
    .
    .
    .
CALC = Z + B
    .
    .
    .
RETURN
    .
    .
    .
END
```

In this example, the value of CALC is computed and its value is returned to the calling statement.

The FUNCTION subprograms are logically terminated during execution by a RETURN statement. They are physically terminated during compilation by an END statement.

The arguments in a FUNCTION statement can be considered as dummy variable names that are replaced at the time of execution by the actual arguments supplied in the function reference in the calling program. The actual arguments must correspond in number, order, and type with the dummy arguments, and can be constants, nonsubscripted variables, subscripted variables, expressions, array names, or other subprogram names.

If the value of a dummy argument is changed by the subprogram, then the actual argument in the calling program is also changed. In this case, the actual argument must be a nonsubscripted variable, a subscripted variable, or an array name. The actual argument cannot be a constant, an expression, or a subprogram name.

Dummy arguments cannot appear in an EQUIVALENCE statement in the FUNCTION subprogram.

Dummy arguments can be subprogram names. The corresponding actual arguments in the calling program must be names that have appeared in an EXTERNAL statement in the calling program. The dummy arguments can also appear in an EXTERNAL statement. In this way, a subprogram name used as an actual argument in a calling program can be passed to a subprogram, which in turn can pass it on to another subprogram.

If a dummy argument is an array name, a DIMENSION statement or a COMMON (with dimensions) statement for that array must appear in the FUNCTION subprogram. Further, the corresponding actual argument must be a dimensioned array name.

If a dummy array name appears in a COMMON statement, then the actual argument in the calling program must be an array name that appears in the identical place in the COMMON statement.

If a dummy variable name appears in a COMMON statement, then the value of the actual argument in the calling program replaces the value of the dummy name in COMMON immediately upon entry to the FUNCTION subprogram.

The SUBROUTINE Statement

The SUBROUTINE statement is used to name SUBROUTINE subprograms and must be the first statement of a SUBROUTINE subprogram. It cannot appear anywhere else in the subprogram nor can it appear in a main program.

General Form. SUBROUTINE *name* ($a_1, a_2, \ldots, a_n$)

name is the name of the subprogram.

$a_1\ a_2, \ldots, a_n$ are the dummy arguments, and can be nonsubscripted variable names, or array names, or the

IBM

FORTRAN CODING FORM

FORM 624-7999

TITLE	PROJECT NUMBER	ANALYST	NO. OF CARDS	SHEET ___ OF ___

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT	IDENTIFICATION
C		PROGRAM FOR FINDING THE LARGEST VALUE	
C		ATTAINED BY A SET OF NUMBERS	
		DIMENSION A(36)	
		READ (1,1)N,(A(I),I=1,N)	
	1	FORMAT (I3/(12F6.2))	
		BIGA=A(1)	
		DO 20 I=2,N	
		IF (BIGA.LT.A(I)) BIGA = A(I)	
20		CONTINUE	
		WRITE (3,2)N,BIGA	
	2	FORMAT (21H1THE LARGEST OF THESE,I3,11H NUMBERS IS,F9.2)	
		STOP	
		END	

Figure 9. Fortran Source Program

Item to Check	*Coding Error*
A-conversion	Field width, w, exceeds the word size, $f + 2$.
Arithmetic expressions	Real and integer numbers, both constants and variables, mixed in invalid combinations. Often, a real constant is written without a decimal point.
DO parameters	Subscripted integer variable or expression used as a parameter.
FORMAT statements	FORMAT specifications and I/O list not compatible.
Fortran language	Misspelled Fortran-language word such as EQUIVALENT instead of EQUIVALENCE.
H-conversion	Incorrect count for n of nH.
Order of source deck	Specification statements or FORMAT statements are out of sequence.
Program flow	Statement transfers into the range of a DO.
	Unreferenced statement after a GO TO, RETURN, OR STOP. END statement encountered in program flow.
Statement numbers	Use of same statement number more than one time.
	Absence of a referenced statement number.
Subprograms	FUNCTION or SUBROUTINE statement missing at beginning of a subprogram; RETURN statement missing; END statement missing.
Subprogram names	Name is same as a variable name used in the program.
SUBROUTINE statement arguments	Dummy arguments that are subscripted or equivalenced variables.
Subscripted variables	Each subscripted variable, including those in lists, does not appear in a DIMENSION statement.
Variables	Variables not defined in a read I/O list or on the left-hand side of an arithmetic statement before being used on the right-hand side of an arithmetic statement or in a CALL statement.

Punching the Source Program

The Fortran statements, prepared as described in *Writing the Source Program,* are normally punched into the standard Fortran card, Form 888157, shown in Figure 10.

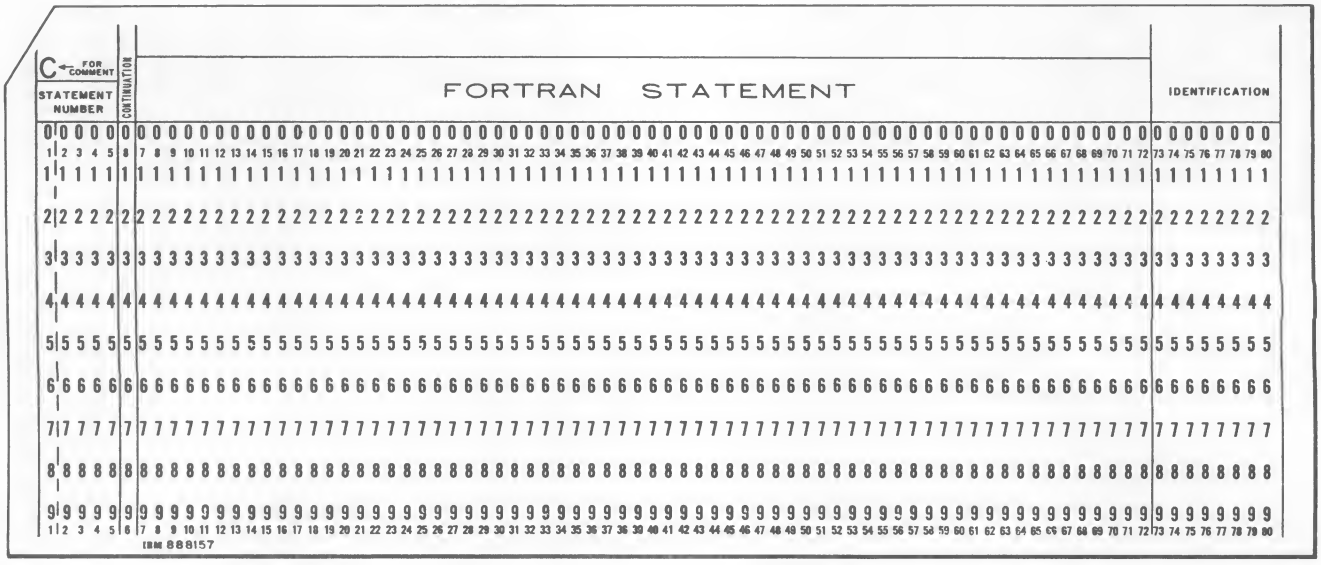


Figure 10. Fortran Source Card

Table of Source Program Characters

The following table indicates the list of characters in character Set H that can be used in a Fortran source program. The equivalent characters in character Set A are also given. The characters are shown in collating sequence.

Set H (Fortran Characters)	*Set A*	*Card Punches*
Blank	Blank	Blank
.	.	12-3-8
)	□	12-4-8
+	&	12
$	$	11-3-8
*	*	11-4-8
—	—	11
/	/	0-1
,	,	0-3-8
(	%	0-4-8
=	#	3-8
'	@	4-8
A	A	12-1
B	B	12-2
C	C	12-3
D	D	12-4
E	E	12-5
F	F	12-6
G	G	12-7
H	H	12-8
I	I	12-9
J	J	11-1
K	K	11-2
L	L	11-3
M	M	11-4
N	N	11-5
O	O	11-6
P	P	11-7
Q	Q	11-8
R	R	11-9
S	S	0-2
T	T	0-3
U	U	0-4
V	V	0-5
W	W	0-6
X	X	0-7
Y	Y	0-8
Z	Z	0-9
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9

Note: The character $ can be used in Fortran only in an H-conversion field.

disk unit 2. The programmer can inform the machine operator of this fact by using a PAUSE card, telling him to ready the drive. The message would be:

PAUSE READY THE PACK ON DISK DRIVE 2.

COPY Card

The COPY card is applicable only to tape-resident systems. It is used when the user wants to duplicate the system tape. The COPY option permits the user to duplicate the SYSTEM file, including the LIBRARY file if it resides on the same tape, on another tape (WORK1). The general format for the COPY card is:

COPY [*any message and/or identification*]

HALT Card

The HALT card indicates to the System Control Program that processing has been completed. It is the last card of a stack. The content of the HALT card is printed on the MESSAGE file. The general format for the HALT card is:

HALT [*any message and/or identification*]

Fortran Processor Program

The Fortran Processor Program is made up of the following:

Fortran compiler
Fortran loader
Fortran library

The following sections contain a description of the various components of the Fortran Processor Program and a description of the output from the components, to include a description of any diagnostic messages that the user may receive as a result of processing operations.

Fortran Compiler

The Fortran compiler is the processing element of the processor program. It operates under control of the System Control Program. The compiler translates source program statements written in the Fortran language into machine-language and interpretative instructions in a relocatable format. These instructions are then acceptable to the Fortran loader.

Instructions in the relocatable format contain symbolic and relative addresses. These addresses are relative to a base address of 001. These symbolic and relative addresses are converted to actual addresses by the Fortran loader. The result of this conversion is an object program in the absolute format.

Relocatable formats permit inclusion of several relocatable programs at load time. Inter-program communication is accomplished by the use of symbolic names, whose corresponding addresses are substituted at load time by the Fortran loader.

Compiling Variables

Five compiling variables are available in the compiler. These variables include:

1. Integer size (the number of significant digits) to be used at object time.
2. Real size to be used at object time.
3. Object machine size.
4. Availability of the multiply/divide feature.
5. Main program name.

These variables can be specified by using compiler option control cards. A description of each of these variables follow.

Integer Size. The assumed integer size is 5. Preceding a compilation, the object-time integer size, *k*, can be specified to be any value from 1 through 20.

Real Size. The assumed real size is 8. Preceding a compilation, the object-time real size, *f*, can be specified to be any value from 2 through 20, where *f* is the mantissa length. The compiler will reserve $f + 2$ positions for each real variable to allow for a 2-digit exponent.

Object Machine Size. The assumed object machine size is 11999. If the object machine is greater than the assumed value, the highest core storage address available at object-time can be specified to serve as a base address for COMMON storage.

Multiply/Divide Feature. The multiply/divide feature is assumed to be available in the object machine. If the feature is not available, this fact can be specified by using a compiler control card.

Main Program Name. The main program (phase) name is assumed to be ///. By using a compiler control card, the user can specify a main program (phase) name that is three alphameric characters in length. The name appears on the first card, disk, or tape

record of the relocatable output generated by the compiler. If an absolute deck is specified to the loader, the three characters are included in the first card of the absolute deck. These three characters are used to identify the program if it is stored as a phase on the SYSTEM file.

If the program (phase) is to be stored on the SYSTEM file as a phase that can be called for execution at any time, the user *must* make sure that the three characters of the program (phase) name are not the same as the name of one of the phases of the System Control Program and/or the Fortran processor. The names of phases of the Fortran processor are in the form *nn*F, where *n* is numeric. The names of the phases of the System Control Program are in the form *xxy*, where *x* is alphabetic and *y* is alphameric. *Appendix II* contains the three-character names of the phases of both the System Control Program and the Fortran Processor Program. Consult the appendix to ascertain that there is no duplication of phase names.

Fortran Compiler Output

The output from the Fortran compiler is on the devices as specified in the ASGN cards. The LIST file output, with an assumed assignment to be the printer, is composed of a source program listing, a name dictionary, and a sequence number dictionary. Any or all of these types of output can be omitted by using an output option control card.

Source Program Listing. A listing of the Fortran source program is output by the Fortran compiler. The listing is made up of input card images and a compiler-generated sequence number. A sequence number appears in front of each card except for a comment card. An example of a source program listing is shown in the sample program included as *Appendix IV*.

Name Dictionary. A name dictionary is made up of the names of simple variables and/or arrays that are included in the source program. Associated with each variable and/or array is the corresponding object-time relocatable address. These addresses are relative to a base address of 001. An array-name address corresponds to the first element of the array. An example of a name dictionary is shown in the sample program included as *Appendix IV*.

Sequence Number Dictionary. The sequence number dictionary is made up of the compiler generated sequence numbers and the corresponding object-time relocatable addresses. These addresses are relative to a base address of 001. The address indicates the position of the first character of the transformed statement.

Sequence numbers do not necessarily appear in order in the dictionary. Further, a sequence number may appear twice in the dictionary. (This could occur in the case of a READ/WRITE statement. A sequence number would represent the actual input/output subroutine call; the same sequence number would represent the transformed I/O list.) Specification statement sequence number addresses appear in the dictionary with the addresses equivalent to another sequence number address.

In addition to output on the LIST file, the user can specify, by way of an OUTPUT ASGN card, that a punched-card deck in the relocatable format be produced by the compiler. This card deck is the same object program that is located on the LOADER file in mass storage. When a punched-card deck is specified, the user has the option of specifying that the relocatable object program on the LOADER file be omitted. (This is accomplished by using an ASGN OMIT card.) Although it is possible to have the object program in the relocatable format on both the LOADER file and the OUTPUT file, generally only one form of output is chosen.

Fortran Compiler Diagnostics

Diagnostic information is produced pertaining to the intelligibility and consistency of the source program as defined by the language specifications section of this publication. If an error is detected by the compiler, a message is printed on the LIST file informing the user of his error. The message printed on the LIST file can have a maximum of three parts.

The first part of the message is a flag that indicates the severity of the error. A single asterisk (*) indicates a diagnostic of a warning type. A single asterisk allows compilation to continue and relocatable output is produced.

Three asterisks (***) indicate a severe error, and compilation is suspended. When this type of error occurs, the message

```
COMPILATION SUSPENDED-REMOVE NECESSARY
CARDS FROM READER
```

is printed on the LIST file. In these instances, control is returned to the System Control Program, and a control card for the next job is read.

The second part of the message is the sequence number of the statement in error. In certain cases, the sequence number is not included as a part of the message. For example, if the name table is being searched and an error occurs, no sequence number is included in the message.

Object Programs

This section contains a description of the way object programs are contained in core storage at the time they are executed. Topics discussed in this section include storage allocation, the standard loader overlay, subprograms, statement expansions, and an explanation of the core-storage allocation for the sample program contained as *Appendix IV*.

Storage Allocation

In general, a Fortran source program is translated into interpretive strings representing arithmetic expressions or input/output lists, sequences of tests for logical expressions, and subroutine calls with necessary parameters.

Each Fortran main program or subprogram is relative to a relocatable base address of 001, and is ordered in the following manner.

1. Constants common to every program and subprogram
2. Arrays
3. Simple variables and constants
4. Executable statement expansions
5. FORMAT statement and input/output list expansions
6. Subprogram prologue and epilogue.

Each real variable requires $f + 2$ positions of core storage. Each integer variable requires k positions of core storage. Each logical variable requires one position of core storage.

The number of core-storage positions required for an array is determined by multiplying the array dimensions together and multiplying that product by the appropriate word size according to type (real, integer, or logical).

Arithmetic constants appear in core storage with a length as written, except leading zeroes are dropped and two positions are used for the exponent of real constants. A maximum of $f + 2$ or k positions are required; however, less may be used. Logical constants require one position of core storage.

Arrays and simple variables in COMMON are assigned absolute (i.e., non-relocatable) addresses beginning with the high core-storage address and proceeding with decreasing addresses. No relocatable load cards are generated.

The Fortran loader normally loads a main program and possibly several subprograms in the relocatable format from the LOADER file. The first program read from the LOADER file is loaded into core storage beginning at address 5701. Subsequent loading follows the preceding program. Any required standard subprogram will then be relocated and loaded following the user-programs. A communication and system-constant area is prepared from core-storage address 950 to 1010.

When loading is complete, the loader prepares for object-program execution by calling a set of standard subprograms, called the standard loader overlay, always required by the object program. Execution then takes place.

Standard Loader Overlay

The standard-loader-overlay package occupies core-storage positions 1010 through 5700 during object-program execution. The overlay package includes routines such as the arithmetic interpreter, object-time error messages, and the general READ/WRITE service routines for initialization, file opening, end-of-file detection, buffer clearing, and unit record (card reader, card punch, printer, and console printer) input/output processing. Also included are the input/output list routine, and several formatting routines to handle FORMAT initialization, FORMAT-list interaction, the slash (record delimiter) element, and I, E, and F input and output conversions. Additional selectively included subroutines are normally required when using a FORMAT statement, as described with character counts in the following section.

Selectively Included Standard Subprograms

A list of standard subprograms that exist on the LIBRARY file in the relocatable format follows. Each is relocated and loaded by the loader, if required by the user-program(s). Note that some standard subprograms require other standard subprograms. The use of certain capabilities of the Fortran language as well as explicit subprogram references require the inclusion of standard subprograms. The external names that are associated with the subprograms appear in the name map when the corresponding subprogram is loaded.

Subprogram	*External Name*	*Character Count*
FORMAT left parenthesis	G.	22
FORMAT internal right parentheses	H.	34
FORMAT scale factor	L.	18
FORMAT H-specification	M.	153
FORMAT L-specification, requires P.	N.	135
FORMAT A-specification, requires P.	O.	191
FORMAT A-, L-, I-, F-, or E-specification	P.	230
FORMAT final right parenthesis	Q.	95

Subprogram	*External Name*	*Character Count*
Random file processing for FIND or random READ/WRITE (disk-resident system only)	A. B. C. D. E. F.	189
Each logical file control word for file *i*, where $0 \leq i \leq 9$	*i	21
Unedited READ/WRITE	I. J. K.	369 (disk) 369 (tape)
BACKSPACE	R.	243 (disk) 163 (tape)
END FILE	S.	153 (disk) 86 (tape)
REWIND	T.	74 (disk) 47 (tape)
FIND (disk-resident system only)	U.	177
Variable file name search	V.	106
Address vector for random file definition vectors, requires V.	X.	33
Address vector for sequential file control words, requires V.	Y.	33
Object-time FORMAT, requires G., H., L., M., N., O., P., and Q.	W.	847
Tape I/O with disk-resident system	Z.	166
Subscripting	)A)B U) V) Y) Z)	283
Multiply/divide subroutine	)C)D)E	471
DO, or implied DO in I/O list	)F)G)H	173
Sense light	SSLITE W) X)	59
Sense light test, requires SSLITE	SLITET	82
Sense switch test	SSWTCH	83
End-of-file test	EOF	44 (disk) 44 (tape)
Phase linkage (also required by STOP statement)	LINK	22 (disk) 22 (tape)

Other uses of the Fortran language, as well as explicit references to standard Fortran functions, cause the loading of an 8-character BCE instruction for each function, plus the coding for the function itself. The external name associated with the BCE instruction is of the form ,*c* , where *c* represents the alphameric character unique to each standard function. The complete set of required BCE instructions for a particular program exists together in core storage after the program(s) have been loaded by the Fortran loader. The external name ,9 appears at the beginning of the BCE instructions.

Subprogram	*External Name*	*Character Count*
QUIT — leave interpretive coding for executable coding	+ ,0	8
Relational expression testing, including arithmetic IF	,0	8
	,1	8
	,2	8
	,3	8
	,4	8
	,5	8
	(0 (1 (2 (3 (4 (5	134
Absolute value — ABS	,A	8
— IABS	,B	8
	(A	11
Positive difference — DIM	,D	8
— IDIM	,V	8
	(D	64
Extreme value — AMAX0, requires ,R and (R	,E	8
— AMAX1	,F	8
— MAX0	,G	8
— MAX1, requires ,I and (I	,H	8
— AMIN0, requires ,R and (R	,J	8
— AMIN1	,K	8
— MIN1, requires ,I and (I	,O	8
— MIN0	,Y	8
	(E (F (G (H (J (K (O (Y	248
Truncation — IFIX	,I	8
	(I	129

Subprogram	*External Name*	*Character Count*
Float — FLOAT	,R	8
	(R	36
Remaindering — MOD	,M	8
	(M	88
— AMOD, requires ,R and (R and ,I and (I	,P	8
	(P	88
Transfer of sign — ISIGN	,U	8
— SIGN	,W	8
	(W	69
Truncation to real — AINT, requires (I and (R	,Z	8
	(Z	28
Complement compression, required by a subprogram with adjustable dimensions — CMCM, requires subscripting	,8	8
	(8	18
Integer exponentiation	,7	8
	(7	311
Power — logarithmic part of real exponentiation, requires A) and (L	,6	8
Cosine — COS, requires A) and (S	,C	8
Sine — SIN, requires A)	,S	8
	(C, (S	437
Common logarithm — ALOG10, requires A)	,L	8
	(L	370
Natural logarithm — ALOG, requires A) and (L	,N	8
Square root — SQRT, requires A)	,Q	8
	(Q	224
Arctangent — ATAN, requires A)	,T	8
	(T	492
Exponential — EXP, requires A)	,X	8
	(X	262
Power series subroutine required by various transcendental functions	A), B), C), D), E), F), G), H), I), J), K), L), M), N), O), P)	287

Statement Expansions

The COMMON, EQUIVALENCE, and type statements do not generate object-time characters. These statements serve only as information to the compiler.

The DATA statement information literals, converted to internal notation, appear in the space allocated for their respective variables or array elements. Either $f + 2$, k, or one position is used.

An arithmetic statement defining a numerical value is translated into a character string consisting primarily of one-character operators and three-character (address) operands. References to standard functions require one character of core storage, plus the inclusion of a BCE for the function and the function itself. Subscripted variable references normally require five characters plus six for each dimension and the inclusion of the subscript subroutine.

An arithmetic statement defining a logical value is normally translated into a sequence of eight character tests on the various logical variables. Arithmetic strings, as described in the preceding paragraph, followed by a four-character relational test (one-character function reference plus a three-character parameter) represent a relation.

A GO TO statement results in a four-character branch instruction. A computed GO TO results in as many eight-character branch instructions as there are statement numbers in the computed GO TO.

A logical or arithmetic IF results in very nearly the same object expansion as an arithmetic *statement, and* can be approximated in essentially the same fashion.

The DO statement generates 23 characters, and causes the inclusion of the DO subroutine. Four characters are required after the final executable statement in the range of the DO.

The CONTINUE statement generates no object characters.

The PAUSE statement generates either one character or five characters, depending upon the absence or presence of the optional integer constant.

A STOP statement generates eleven object-time characters and causes the inclusion of the LINK subroutine.

The END statement generates five object-time characters.

Input/output lists are translated into a sequence of subroutine calls and interpretive strings. The sample program shown in *Appendix IV* gives an example of input/output lists.

A FORMAT statement is translated into a sequence of subroutine calls with parameters corresponding to the FORMAT elements used. Additional selectively included standard subprograms are normally required.

A READ or WRITE statement, or one of the manipulative input/output statements is translated to a subroutine call with required parameters. Corresponding lengths are:

READ/WRITE	19
FIND	10
END FILE	7
REWIND	7
BACKSPACE	7

A standard subprogram is required for each type of manipulative input/output statement used. An expression consisting of something more complex than a single non-subscripted variable or constant in a FIND statement, or in the random form of a READ or WRITE statement, causes the generation of additional characters, similar to the arithmetic statement. If a variable file name is used, ten additional characters are required for each reference, as well as two or three standard subprograms.

The DEFINE FILE statement causes the generation of seventeen characters for each file defined. The external names $\$i$, where $0 \leq i \leq 9$, are also generated by the DEFINE FILE statement.

The FUNCTION and SUBROUTINE statements cause generation of a prologue for the evaluation of parameters and array dimension calculations, if required, and an epilogue to reset any values required and return control to the proper place.

The RETURN statement generates four characters.

The CALL statement generates four characters, plus three characters for each actual argument. If the actual arguments consist of more complicated expressions than a single non-subscripted variable, constant, or alphameric field, then an expression-evaluation string for either arithmetic or logical expressions is also generated.

It was found that twenty-three typical programs had statement expansions averaging approximately thirty characters per statement. A program containing numerous long FORMAT statements had a significantly higher average, whereas a program almost entirely composed of simple arithmetic statements had a somewhat lower average. In all cases, additional storage was required for arrays, variables and constants, and required subprograms. Thus, the total core-storage requirement above the base loading point of 5700 varied considerably, ranging from about 500 characters to about 10,000 characters.

Core-Storage Allocation for the Sample Program

The discussion in this section refers directly to the sample program that is shown as *Appendix IV*. Core storage has been allocated for the object program in the following manner.

Area	*Content*
0-949	Resident System Control Program functions
950-1009	Communication area, system constants
1010-5700	Standard loader overlay
5701-6201	Main program, relocated
5701-5850	Constants, arrays, variables
5851-5987	Executable and END statements
5988-6201	FORMAT's, input/output lists
6202-7908	Selectively included standard subprograms
6202-6223	LINK subroutine, required by STOP statement
6224-6506	Subscripting subroutine
6507-6977	Multiply/divide subroutine
6978-7150	DO subroutine
7151-7198	Function branches for relational functions, required by logical IF statement
7199-7332	Relational expression testing
7333-7354	FORMAT left parenthesis
7355-7388	FORMAT internal right parenthesis
7389-7541	FORMAT H-specification
7542-7771	FORMAT A-, L-, I-, F-, or E-specification
7772-7866	FORMAT final right parenthesis
7867-7887	File control word for logical file 1
7888-7908	File control word for logical file 3

This accounts for all external names appearing in the name map, recognizing that the main program name is the assumed value ///.

The statement expansion area occupies 351 characters, averaging approximately 35 characters per statement. This average includes the END statement, but excludes the DIMENSION statement. Note that the FORMAT expansion required considerably more than the average number of characters, whereas the arithmetic statement, sequence number 004, required less.

The location and length of each statement expansion can be determined by adding the program base loading point to the relative addresses shown in the sequence number dictionary. The program base loading point for the first program that is loaded is 5700. Subsequent programs are loaded immediately following the first program. Consequently, their base loading points are higher than 5700.

Jobs

The Fortran system performs two major operations.

1. Translates source programs into object programs.
2. Starts the execution of object programs.

Because these operations are performed by the Fortran processor part of the system, the operations are called processor jobs.

Two other operations, maintaining the Fortran subprogram library and updating the Fortran system, are also considered jobs. Maintaining the Fortran subprogram library is called a library job. Updating the Fortran system is called an update job. Update jobs are described in *Updating a Fortran System.*

Under control of the System Control Program, it is possible to perform one or more jobs without operator intervention. This process is called stack processing. If the system resides on disk, or if the system resides on tape unit 2, 3, 4, 5, or 6, a stack is *always* made up of the *card-boot* deck, a SYSTEM ASGN card, the particular job(s) to be performed, and a HALT card. If the system resides on tape unit 1, a stack is made up of the particular job(s) to be performed and a HALT card. (Pressing the tape load key serves the same function as the *card boot* and the SYSTEM ASGN card when the system tape is on unit 1.)

In performing a job, the following factors must be taken into consideration.

1. The kind of input for the job.
2. The use of the logical files.
3. The machine-operator procedures to be followed.

The kinds of input for processor jobs and library jobs are discussed in the following sections (*Preparing Processor Jobs* and *Preparing Library Jobs*).

The general use of logical files is discussed in *Logical Files.*

In most cases, the user does not need to be concerned about the logical files because the Fortran system defines the files and assigns them to specific input/output devices. In the description of preparing processor jobs that follows, any file assignment that the user must make is explained.

The machine-operator procedures to be followed are described in *Performing Jobs.*

Preparing Processor Jobs

This section describes each processor job. They are:

FORTRAN RUN
LOADER RUN
PRODUCTION RUN

Each processor job description includes:

1. *Assumed input device.* This entry refers to the device on which the input is assumed to be located. For the 1402, READER 1 means that the cards are selected into stacker 1. For the 1442, READER 1 means unit 1.
2. *Input.* This entry refers to the type of input for the job.
3. *Assumed output devices.* This entry refers to the device(s) on which the output is assumed to be located. For the 1403, PRINTER 2 means that 132 print positions are available. For the 1443, PRINTER 2 means that 144 print positions are available. For the 1402, PUNCH 4 means that the cards are selected into stacker 4. For the 1442, PUNCH 1 means unit 1.
4. *Output.* This entry refers to the type of output that the user *always* gets as a result of the job, unless specified otherwise.
5. *Output options available.* This entry refers to the type of output the user can get by using output option cards.
6. *Required user assignments.* This entry describes any additional logical file assignments that the user must make to perform the job.
7. *Control cards.* This entry describes the method of punching any required control cards and output option cards.
8. *Arrangement.* This entry references a figure that shows the manner of arranging card input for the job.

NOTES.
1. Any logical file assumed assignment can be changed. If the user wishes, he can change the assignment by using an ASGN card.
2. NOTE and PAUSE cards can be placed between, but not within, job decks.

FORTRAN RUN

This is the type of run that translates a source program written in the Fortran language into an object program in the relocatable format. The output for this run is then ready for processing by the Fortran loader.

Assumed Input Device. INPUT file on READER 1.

Input. Source program.

Assumed Output Devices. MESSAGE file on PRINTER 2; LIST file on PRINTER 2; LOADER file on 1311 or 1301 UNIT 0, START 010000, END 012000 or TAPE UNIT 3.

Output.

1. Source-statement diagnostics on the LIST file, if errors are sensed.
2. Source program listing on the LIST file, unless an output option control card specifies that the listing be omitted.
3. Name dictionary on the LIST file, unless an output option control card specifies that the name dictionary be omitted.
4. Sequence number dictionary on the LIST file, unless an output option control card specifies that the sequence number dictionary be omitted.
5. Object program in the relocatable format on the LOADER file, unless the LOADER file has been omitted.

Output Options Available. Object program in the relocatable format on the OUTPUT file.

Required User Assignments. If the object program is to be written on the LOADER file, no file assignments are required. If the user wants the object program on the OUTPUT file, an OUTPUT ASGN card is required.

Control Cards.

1. The RUN card is the only required control card. Punch the RUN card in the following manner.

Columns	*Contents*
6-12	FORTRAN
16-18	RUN

2. If the object program is to be on the OUTPUT file, an OUTPUT ASGN card is required. This card must precede the RUN card. Generally, the only time that the user would want to use the OUTPUT file is when a punched card deck in the relocatable format is desired. If this is the case, punch the OUTPUT ASGN card in the following manner:

Columns	*Contents*
6-11	OUTPUT
16-19	ASGN
21-27	PUNCH *n*

If the OUTPUT file is assigned to PUNCH *n*, *n* can be 0, 4, or 8 for 1402, 1 or 2 for 1442, 3 for 1444.

If the OUTPUT file is assigned, the user may wish to omit the LOADER file. If this be the case, a LOADER ASGN card is required and must precede the RUN card. Punch the LOADER ASGN card in the following manner.

Columns	*Contents*
6-11	LOADER
16-19	ASGN
21-24	OMIT

3. The following cards are output option control cards and compiler option control cards. They are used when any of the options are desired. The option cards immediately precede the Fortran source program in the INPUT file and can be in any order. Option control cards are output on the LIST file.

a. If the integer size is to differ from the assumed value of 5, punch the following card. *Note:* $01 \leq nn \leq 20$.

Columns	*Contents*
1-8	$INTEGER
10-13	SIZE
15	=
17-18	*nn*

b. If the real size is to differ from the assumed value of 8, punch the following card. *Note:* $02 \leq nn \leq 20$.

Columns	*Contents*
1-5	$REAL
7-10	SIZE
12	=
14-15	*nn*

c. If the object machine differs from the assumed value of 11999, punch the following card. *Note:* $11999 \leq nnnnn \leq 15999$.

Columns	*Contents*
1-7	$OBJECT
9-15	MACHINE
17-20	SIZE
22	=
24-28	*nnnnn*

d. If the multiply/divide feature is not present in the object machine, punch the following card. (The multiply/divide feature is assumed by the Fortran compiler to be available.)

Columns	*Contents*
1-3	$NO
5-12	MULTIPLY
14-19	DIVIDE

when the user specifies that an absolute deck be punched as a result of a LOADER RUN.

NOTE. If the SYSTEM file resides on tape, the user must specify that the phase be inserted after a particular phase on the SYSTEM file. In order that this be accomplished, punch the three-character phase name after which the phase is to be inserted in columns 21-23 of the UPDAT control card, the first card of the absolute deck. (Columns 21-23 of the UPDAT card punched by the LOADER contains the name of the phase that is to be inserted and must be changed.) 90F is the name of the last phase of the Fortran system. When deleting a phase and inserting another phase in its place, it is recommended that these two user-update jobs be performed in two separate stacks.

2. If a phase is to be deleted from the SYSTEM file, punch the following card.

Columns	*Contents*
6-15	[*any user comments*]
16-20	UPDAT
21-23	*three-character phase name*
24	comma
25-30	DELETE

In the control card, *three-character phase name* refers to the three-character main program name. The phase name appeared in columns 21-23 of the absolute deck that was used to insert the phase on the SYSTEM file.

Arrangement. The arrangement of input cards for a user-update job is shown in Figure 21.

NOTE. If the system is disk-resident and the capacity of the SYSTEM file is exceeded when a user-update job is performed, Halt 629 (unequal address compare) occurs. A phase on the SYSTEM file that is no longer used that is as large or larger than the phase to be inserted can be deleted to permit the new phase to be inserted in its place.

Preparing Library Jobs

Library jobs are associated with the maintenance of the Fortran library. The Fortran library is a mass-storage file that supports the Fortran loader. The file contains a library table (disk-resident systems only) and subprograms, such as standard Fortran functions and subroutines.

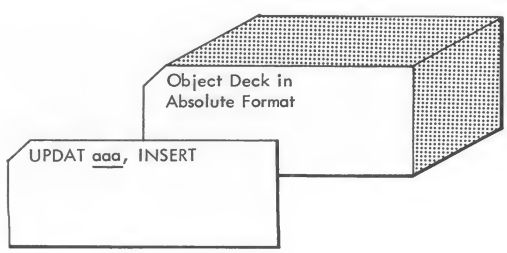


Figure 21. User-Update Job

The three standard library jobs are:

1. Library build that enables the user to *define* a LIBRARY file. A library-build job, performed when a disk-resident Fortran system is built, defines a LIBRARY file on the same disk unit as the SYSTEM file. The limits of this LIBRARY file are 012000 *through* 013899. Thus, the assumed assignment is 1311 or 1301 UNIT 0, START 012000, END 013900.

   After the library-build job has been performed, the LIBRARY file contains the library table. The library table is thirty sectors in length. The user can enlarge the name table according to his needs.

   If the system is tape resident, the library is already built. As a result, the tape user need be concerned only with library changes and library listings.
2. Library listing that enables the user to get a list of the library subprograms and/or the names of the subprograms that are in the LIBRARY file.
3. Library change that enables the user to modify the content of a LIBRARY file. A library-change job, first performed when the disk-resident system is built, transfers the subprograms to the LIBRARY file after the file has been defined.

At the completion of a library job (LIBRARY RUN) on a disk-oriented system, three messages are printed on the LIST file. The messages are:

END OF LIBRARY RUN
LIBRARY ASSIGNED *nnnnnn* TO *nnnnnn*
REMAINING SECTORS *nnnnnn* TO *nnnnnn*

In the message, *nnnnnn* is a disk address. From these messages, the user is able to determine the size of the present library, and the number of sectors available for any additional subprograms that may be added.

Any subprogram is stored in disk storage one card per sector in the move mode. As the input for a LIBRARY RUN must be in card form, the user can determine whether a subprogram will fit in the library by merely counting the cards in the deck output from a FORTRAN RUN.

If the LIBRARY file is full and the user wants to add a new subprogram, one of two steps can be followed.

1. The user can define and build a new LIBRARY.
2. The user can delete an existing subprogram from the LIBRARY file and insert the new subprogram. This can be done if the new subprogram will occupy the same number or fewer sectors than the old subprogram.

For tape-oriented systems, at the completion of a LIBRARY RUN, the message,

END OF LIBRARY RUN

is printed on the LIST file.

Library Build

Library-build jobs apply only to disk-resident systems. Each library-build job defines a LIBRARY file that contains a name table 30 sectors in length. If a table of more (or less) than 30 sectors is required, specify the sector number desired in the control card.

The control cards required for a library-build job are:

1. A LIBRARY ASGN card is required if the assignment of the LIBRARY file differs from that assumed by the System Control Program. Punch the ASGN card in the following manner:

Columns	*Contents*
6-12	LIBRARY
16-19	ASGN
21-57	1311 UNIT *n*, START *nnnnnn*, END *nnnnnn* *or* 1301 UNIT *n*, START *nnnnnn*, END *nnnnnn*

   For disk, the value *n* indicates the number of the disk unit, and can be 0, 1, 2, 3, or 4; *nnnnnn* represents a disk address. The limits of the library must be supplied.

2. Punch the required RUN card in the following manner:

Columns	*Contents*
6-12	LIBRARY
16-18	RUN

3. Punch the library-build card in the following manner:

Columns	*Contents*
16-20	BUILD
21-23	[*nnn*]

   The value *nnn* is used only when the name table is to differ from 030 sectors.

4. The END card must be the last card of a library-build job. Punch the END card in the following manner:

Columns	*Contents*
16-18	END

The arrangement of control cards for a library build job is shown in Figure 22. The cards are read from the CONTROL file.

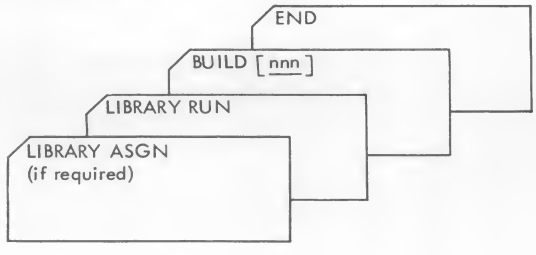


Figure 22. Library Build

Library Listing

The user can request three types of library listings.

1. A listing of the names or headers of all the subprograms in the Fortran library.
2. A listing of the entries in a specific subprogram.
3. A listing of the entries in every subprogram.

The control cards required for a library-listing job are:

1. A LIBRARY ASGN card is required if the assignment of the library file differs from that assumed by the System Control Program. See *Library Build* for the format of the ASGN card.

2. The required RUN card is punched in the following manner.

Columns	*Contents*
6-12	LIBRARY
16-18	RUN

3. One of the following three cards are required for the library-listing job. The one that is selected depends upon the type of listing that is required.

   a. If a listing of the headers of all the tape subprograms is required, punch the following card. If a listing of the names and disk addresses of all the disk subprograms is required, punch the following card. The listing is output on the LIST file.

Columns	*Contents*
16-19	LIST
21-27	HEADERS

   b. If a listing of the entries in a specific subprogram is required, punch the following card. The listing is output on the LIST file.

Columns	*Contents*
6-11	*name*
16-19	LIST

   name is the six-character name of the specific subprogram entries that are required.

   c. If a listing of the entries in every subprogram is required, punch the following card. The listing is output on the LIST file.

Columns	*Contents*
16-19	LIST

4. The END card must be the last card of a library-listing job. Punch the END card in the following manner.

Columns	*Contents*
16-18	END

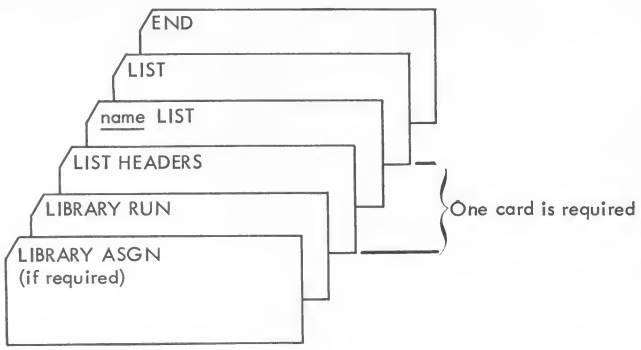


Figure 23. Library Listing

The arrangement of control cards for a library-listing job is shown in Figure 23. The cards are read from the CONTROL file.

Library Change

Fortran subprograms, supplied by IBM or developed by the user, can be added, modified, or deleted.

IBM provides a change deck whenever IBM-supplied standard subprograms should be modified. The change deck includes a LIBRARY RUN card, INSER and/or DELET cards, an END card, and cards containing the changes to be made.

User-change cards must be in the relocatable format, the result of processing by a FORTRAN RUN. In addition to the change cards, the following control cards are required for a library change.

1. A LIBRARY ASGN is required if the assignment of the LIBRARY file differs from that assumed by the System Control Program. See *Library Build* for the format of the ASGN card.
2. The required RUN card is punched in the following manner.

Columns	*Contents*
6-12	LIBRARY
16-18	RUN

3. If a subprogram is to be inserted, punch the following card.

Columns	*Contents*
16-20	INSER

If the system is tape-oriented, the new subprogram is inserted after the last subprogram in the LIBRARY file.

If the system is disk-oriented, the new subprogram is inserted in any available area in the LIBRARY file.

If a subprogram by the same name already exists in the LIBRARY file, it is deleted before the new subprogram is inserted.

When the system is tape-oriented, the WORK1 file is used in conjunction with the LIBRARY file when a subprogram is deleted. Perform a library-copy job to transfer the library from the WORK1 file to the LIBRARY file. See *Library Copy*.

4. If a subprogram is to be deleted, punch the following card.

Columns	*Contents*
6-11	*name*
16-20	DELET

name is the six-character name of the subprogram to be deleted.

All DELET cards must precede all INSER cards. When the system is tape oriented, the WORK1 file is used in conjunction with the LIBRARY file when a subprogram is deleted. Perform a library-copy job to transfer the library from the WORK1 file to the LIBRARY file. See *Library Copy*.

5. The END card must be the last card of a library-change job. Punch the END card in the following manner.

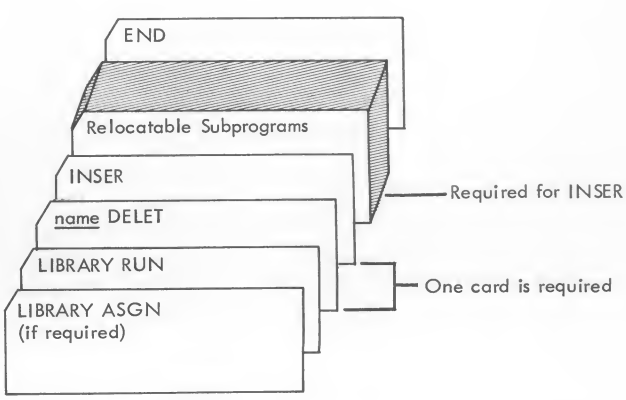
Columns	*Contents*
16-18	END

The arrangement of the control cards and the input for a library change is shown in Figure 24. The control cards and the input cards are read from the CONTROL file.

If the input for a library-change job contains a card that is not recognized by the system, a halt occurs. The message

CARD NOT RECOGNIZED-BYPASS-CONTINUE INSERTION

is printed on the LIST file. In order to continue processing, press START.



● Figure 24. Library Change

Library Copy

The library-copy job is applicable only when the LIBRARY file resides on tape. This job is normally performed immediately after a LIBRARY RUN.

When a subprogram is inserted in place of a subprogram having the same name on the LIBRARY file, or when a subprogram is deleted from the existing library, the WORK1 file is used in conjunction with the LIBRARY file. At the completion of the insertion or deletion, the new or revised library is present on the WORK1 file.

To transfer the library to the LIBRARY file from WORK1, perform a library-copy job.

The following cards are required for a library-copy job.

1. A LIBRARY ASGN card is required if the assignment of the LIBRARY file differs from that assumed by the System Control Program. See *Library Build* for the format of the ASGN card.
2. The required RUN card is punched in the following manner.

Columns	*Contents*
6-12	LIBRARY
16-18	RUN

3. The required COPY card is punched in the following manner.

Columns	*Contents*
16-19	COPY

4. The END card must be the last card of a library-copy job. Punch the END card in the following manner.

Columns	*Contents*
16-18	END

The arrangement of the control cards is shown in Figure 25. The cards are read from the CONTROL file.

Changing File Assignments

Each logical file defined by the Fortran system, with the exception of the SYSTEM file, is assigned to a specific input/output device by the System Control Program. One set of logical file assignments applies to compilation (FORTRAN RUN). A second set of logical file assignments applies to execution (LOADER RUN and PRODUCTION RUN). These assignments can be temporarily changed by using ASGN cards.

Any assignment made by the user remains in effect until:

1. An ASGN card is sensed for the particular logical file, or
2. An INIT (initialize System Control Program assignments) card is sensed, or
3. A HALT card is sensed, signifying the end of a stack.

Preparing ASGN Cards

ASGN cards enable the user to change file assignments for one or more jobs in a stack. The general format for an ASGN card is:

file-name ASGN { *device* / OMIT }

The *file-name* is the specific logical file; *device* is the input/output unit and/or area to which the logical file is assigned.

The assumed file assignments and ASGN card formats relating to specific files are shown in Figure 26. Valid device entries are shown in Figure 27.

ASGN cards are coded in the Autocoder format. When coding ASGN cards, the user must:

1. Leave blanks between items in the operand field as shown in Figure 26. If, for example, the OUTPUT file is to be assigned to disk area 004000 through 004799 on 1301 unit 1, the user would code the ASGN card for punching as shown in Figure 28. The END address that is coded is the address of the next available sector, not the address of the last sector to be used.
2. Left-justify entries in the label, operation, and operand fields, as shown in Figure 28.

File Considerations

SYSTEM File. If the SYSTEM file resides on 1311, drive 0 should be on-line because the System Control Program's assumed assignments are on drive 0. If drive 0 is not on-line, the user must use ASGN cards to change the assumed assignments for the LIBRARY, LOADER, WORK1, WORK2, and WORK3 files.

CONTROL and INPUT Files. If both the CONTROL and INPUT files are assigned to a reader, the assignments must be identical. For example, if the system

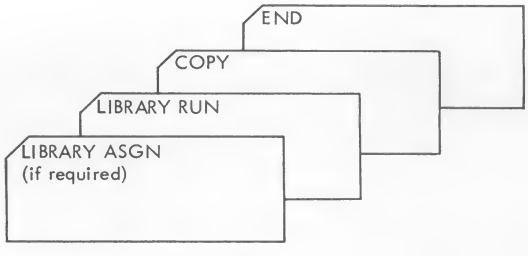


Figure 25. Library Copy

ASGN Card Formats			Assumed Assignment		Remarks
Label Field (Columns 6-15)	Operation Field (Columns 16-20)	Operand Field (Columns 21-72)	Compilation (FORTRAN RUN)	Execution (LOADER RUN or PRODUCTION RUN)	
SYSTEM	ASGN	{ 1311 UNIT n / 1301 UNIT 0 / TAPE UNIT n }	1311 unit: user-assigned 1301 unit: must be assigned to UNIT 0 Tape unit: user-assigned	1311 unit: user-assigned 1301 unit: must be assigned to UNIT 0 Tape unit: user-assigned	If the system residence is 1311 or 1301, the SYSTEM ASGN card is the only required ASGN card. It must follow the Card Boot in a stack of jobs. Any other SYSTEM ASGN cards in the stack are flagged and bypassed. If the user desires that the Fortran system use less than the number of core storage positions available in the processor machine, punch a comma in column 32 and 12K or 16K beginning in column 34. If the system residence is tape and the tape unit is 1, neither the Card Boot nor the SYSTEM ASGN card is required. (Pressing the TAPE LOAD key achieves the same purpose as the Card Boot and the SYSTEM ASGN card.) If the unit is 2, 3, 4, 5, or 6, both the Card Boot and the SYSTEM ASGN card are required to start systems operations.
CONTROL	ASGN	{ READER n / CONSOLE PRINTER }	READER 1	READER 1	If the CONTROL file and the INPUT file are assigned to the card reader, the assignment must be to the same card reader.
MESSAGE	ASGN	{ PRINTER n / CONSOLE PRINTER }	PRINTER 2	PRINTER 2	When the MESSAGE file is assigned to the CONSOLE PRINTER, carriage control characters used with the 1403 or 1443 printer may appear in the message. If the MESSAGE file and the LIST file are assigned to the printer, the assignment must be to the same printer. The MESSAGE file is equivalent to Fortran file 0.
LIST	ASGN	{ PRINTER n / 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	PRINTER 2	PRINTER 2	If the LIST file and the MESSAGE file are assigned to the printer, the assignment must be to the same printer. The LIST file is equivalent to Fortran file 3.
INPUT	ASGN	{ READER n / 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	READER 1	READER 1	If the INPUT file and the CONTROL file are assigned to the reader, the assignment must be to the same reader. The INPUT file is equivalent to Fortran file 1.
OUTPUT	ASGN	{ PUNCH n / 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	OMIT	PUNCH 4 (1401 and 1460 systems) PUNCH 1 (1440 systems)	The OUTPUT file is equivalent to Fortran file 2.
LIBRARY	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 012000, END 013900 1301 UNIT 0, START 012000, END 013900 TAPE UNIT 1	1311 UNIT 0, START 012000, END 013900 1301 UNIT 0, START 012000, END 013900 TAPE UNIT 1	1311 is assumed if the SYSTEM file is assigned to 1311. 1301 is assumed if the SYSTEM file is assigned to 1301. Tape is assumed if the SYSTEM file is assigned to tape. At execution time (LOADER RUN), the LOADER file can be assigned to READER n. If the MESSAGE, LIST, and WORK5 files are assigned to a printer, the assignment must be to the same printer. The WORK1 file is equivalent to Fortran file 4. The WORK2 file is equivalent to Fortran file 5. The WORK3 file is equivalent to Fortran file 6. The WORK4 file is equivalent to Fortran file 7. The WORK5 file is equivalent to Fortran file 8. The WORK6 file is equivalent to Fartran file 9.
LOADER	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / READER n / OMIT }	1311 UNIT 0, START 010000, END 012000 1301 UNIT 0, START 010000, END 012000 TAPE UNIT 3	1311 UNIT 0, START 010000, END 012000 1301 UNIT 0, START 010000, END 012000 TAPE UNIT 3	
WORK1	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 006400, END 009600 1301 UNIT 0, START 006400, END 009600 TAPE UNIT 4	1311 UNIT 0, START 006400, END 007200 1301 UNIT 0, START 006400, END 007200 TAPE UNIT 4	
WORK2	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 006400, END 009600 1301 UNIT 0, START 006400, END 009600 TAPE UNIT 5	1311 UNIT 0, START 007200, END 008000 1301 UNIT 0, START 007200, END 008000 TAPE UNIT 5	
WORK3	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 006400, END 009600 1301 UNIT 0, START 006400, END 009600 TAPE UNIT 4	1311 UNIT 0, START 008000, END 008500 1301 UNIT 0, START 008000, END 008500 TAPE UNIT 6	
WORK4	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	OMIT	1311 UNIT 0, START 008500, END 009000 1301 UNIT 0, START 008500, END 009000 OMIT for tape systems	
WORK5	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / PRINTER n / OMIT }	OMIT	1311 UNIT 0, START 009000, END 009500 1301 UNIT 0, START 009000, END 009500 OMIT for tape systems	
WORK6	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	OMIT	1311 UNIT 0, START 009500, END 010000 1301 UNIT 0, START 009500, END 010000 OMIT for tape systems	

● Figure 26. ASGN Card Formats and Assumed Assignments

Device Entry and Values of n and nnnnnn	Remarks
{ 1311 / 1301 } UNIT n, START nnnnnn, END nnnnnn n is the number of the disk unit, and can be 0, 1, 2, 3, or 4. nnnnnn is a disk address.	The END address is the address of the next available sector. The values of nnnnnn must adhere to the following rules: 1. WORK1 and WORK2 Files. If the disk unit is 1311, the START address must be a multiple of 200. If the disk unit is 1301, the START address must be a multiple of 800. For both 1311 and 1301, the END address must be a multiple of 40. 2. WORK3, WORK4, WORK5, and WORK6 Files. For both 1311 and 1301, the START and END addresses must be multiples of 10. 3. LIBRARY File. For both 1311 and 1301, the START and END addresses must be multiples of 20. If these rules are violated, the system automatically narrows in the disk area to an area that does adhere to these rules.
TAPE UNIT n n is the number of the tape unit, and can be 1, 2, 3, 4, 5, or 6.	
READER n For 1402, n can be 0, 1, or 2. For 1442, n can be 1 or 2.	For 1402, n represents the pocket into which the cards are stacked. For 1442 or 1444, n represents the number of the unit.
PUNCH n For 1402, n can be 0, 4, or 8. For 1442, n can be 1 or 2. For 1444, n must be 3.	
PRINTER n n can be 1 or 2.	n represents the number of print positions available on the 1403 or 1443. For 1403, a 1 indicates 100 positions and a 2 indicates 132 positions. For 1443, a 1 indicates 120 positions and a 2 indicates 144 positions.
CONSOLE PRINTER	The console printer must be an IBM 1447 without a buffer feature.
OMIT	Select this option when the file is not to be used by the Fortran system. The LIST, OUTPUT, LOADER, WORK4, WORK5, and WORK6 files can be omitted.

Figure 27. Valid Device Entries

is a 1440 and the CONTROL file is assigned to READER 1, *the* INPUT file must also be assigned to READER 1.

MESSAGE and LIST Files. If both the MESSAGE and LIST files are assigned to a printer, the assignment must be identical. For example, if the system is a 1401 and the MESSAGE file is assigned to PRINTER 2, the LIST file must also be assigned to PRINTER 2.

WORK1, WORK2, and WORK3 Disk Files. With disk systems, seek time is the most important factor affecting input/output operations. Therefore, it would be expedient for the user with a multi-unit system to distribute the Fortran files to all of the units, thus making a significant reduction in seek time.

Because the WORK1, WORK2, and WORK3 files handle large amounts of data, inefficient use of the files results in an increase in Fortran time requirements. For the benefit of the single disk unit user, WORK1, WORK2, and WORK3 are handled in a special way. The special way is to assign them to the same area of the disk unit as shown in Figure 29. When this is done, the System Control Program "splits" each cylinder, causing WORK1 to occupy the upper half of each cylinder and WORK2 to occupy the lower half of each cylinder. WORK3 initially occupies the upper half,

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45 50 55 60
0 1	OUTPUT	ASGN	1301, UNIT 1, START 004000, END 004800
0 2			
0 3			

Figure 28. Coding for an OUTPUT ASGN Card

Building a Fortran System

After all sets of cards have been labeled and those sets of cards not applicable to the user's system have been removed, the user is ready to use the prepared system deck to build the Fortran system.

Figure 35 is a block diagram showing the building of a disk-resident system.

The system unit must be prepared for writing the complete system from cards. The user must clear disk unit 0 in the move mode from 000000 to 000199, in the load mode from 000200 to 000259, in the move mode from 000260 to 000299, in the load mode from 000300 to 006399, and in the move mode from 006400 to 019979. The Clear Disk Storage Utility program applicable to the user's system can be used for this operation. As header labels are to be deleted, the write-address mode switch will initially be set off.

Figure 36 shows the disk storage allocation on the system unit.

The control cards for the utility program must be punched in the following manner.

For 1311,

Columns	*Contents*
1-15	M00000000019900
21-35	L00020000025900
41-55	M00026000029900

Columns	*Contents*
1-15	L00030000639900
21-35	M00640001997900

For 1301,

Columns	*Contents*
1-15	M000000000199++
21-35	L000200000259++
41-55	M000260000299++

Columns	*Contents*
1-15	L000300006399++
21-35	M006400019979++

The time required to clear the disk unit in the specified modes is approximately five minutes.

Write File-Protected Addresses

The last card in the section labeled WRITE FILE PROTECT is a control card that is partially prepunched. It is by the use of this control card that the limits of the file-protected area reserved for the SYSTEM file in the disk-storage unit are supplied. The user must indicate in the control card whether the system is to reside on a 1301 or 1311 disk unit. For both the 1301 and 1311, the system must be built on unit 0. In the case of the 1311, the system pack can be used on any drive once the system has been built. The control card is punched as follows:

Columns	*Contents*
1-15	FILE-PROTECT ON (prepunched)
17-20	1301 or 1311
22	0 (prepunched)
24-42	FROM NORMAL ADDRESS (prepunched)
44-49	002500 (prepunched)
51-52	TO (prepunched)
54-59	006400 (prepunched)

File	Mode	File-Protected	Sector Range
SYSTEM File			
Not used	Move	No	000000-000199
	Load	No	000200-000259
	Move	No	000260-000299
	Load	No	000300-002499
System Control Program	Load	Yes	002500-002904
Fortran Processor Program	Load	Yes	002905-002979
Not used	Load	Yes	002980-002999
System Control Program	Load	Yes	003000-003175
Fortran Processor Program	Load	Yes	003176-004200
Area for User's Fortran Object Program Library	Load	Yes	004201-006399
WORK Files	Move	No	006400-009999
LOADER File	Move	No	010000-011999
LIBRARY File	Move	No	012000-013899

● Figure 36. Disk Storage Allocation

After columns 17-20 have been punched by the user, the card must be replaced as the last card of the section.

To use the section when the system is to reside on 1311:

1. Ready the pack on disk drive 0.
2. Set the write-address mode switch on.
3. Set the write-disk switch on.
4. Set the I/O check stop switch on.
5. Press CHECK RESET and START RESET.
6. Place the *write file-protected addresses* section in the card reader.
7. Load the program.
   a. IBM 1402 Card Read-Punch: Press LOAD.
   b. IBM 1442 Card Reader: Press START on the reader, and PROGRAM LOAD on the console.
8. When the system attempts to read the last card,
   a. IBM 1402 Card Read-Punch: Press START.
   b. IBM 1442 Card Reader: Press START on the reader.
9. At the end of the job, set the write-address mode switch off.

To use the deck when the system is to reside on 1301:

1. Set the write-address mode switch on.
2. Set the write-disk switch on.
3. Set the I/O check stop switch on.
4. Press CHECK RESET and START RESET.
5. Place the *write file-protected addresses* section in the card reader.
6. Load the program.
   a. IBM 1402 Card Read-Punch: Press LOAD.
   b. IBM 1442 Card Reader: Press START on the reader, and PROGRAM LOAD on the console.
7. When the system attempts to read the last card,
   a. IBM 1402 Card Read-Punch: Press START.
   b. IBM 1442 Card Reader: Press START on the reader.
8. At the end of the job, set the write-address mode switch off.

The time required to perform this job is approximately 1 minute. The following halts can occur when writing file-protected addresses.

Halt Number (A-Address Register)	*Meaning*
020	Last card condition was sensed before the control card. The control card containing the initial and terminal addresses of the area to be file-protected must be the last card of the deck. When the system is restarted by pressing START, a read operation is performed.
021	An invalid disk type is specified in the control card. 1301 or 1311 are the only valid entries for columns 17-20 of the control card. When the system is restarted by pressing START, a read operation is performed.
022	An invalid disk unit is specified in the control card. The only valid entry for column 22 of the control card is 0. When the system is restarted by pressing START, a read operation is performed.
023	An invalid start address (columns 44-49) is specified in the control card. The start address must be 002500. When the system is restarted by pressing START, a read operation is performed.
024	An invalid end address (columns 54-59) is specified in the control card. The end address must be 006400. When the system is restarted by pressing START, a read operation is performed.
025	Disk unit 0 is not ready. When the system is restarted by pressing START, the disk I/O operation is retried.
026	The area specified in the control card is already file-protected (all or in part). If the system is restarted by pressing START, the entire specified area will be file-protected *and* cleared.
027	The area specified in the control card has neither the "normal" disk addresses (000000-?) nor file-protected addresses. This is a hard halt.
028	Parity check or wrong-length record error occurred on the disk unit while writing addresses. When the system is restarted by pressing START, the disk I/O operation is retried.
029	Parity check or wrong-length record error occurred on the disk unit while determining the existing addressing scheme. This is a hard halt.
030	End of the job.

System Control Card Build

The last card in the section labeled CARD BUILD is a control card that is partially prepunched. It is by the use of this control card that disk residence is determined.

The user must indicate in the control card whether the system is to reside on a 1301 or 1311 disk unit. The assumed disk unit number is 0.

Input	Object program in the relocatable format on the LOADER file
Output	1. Messages to the machine operator on the MESSAGE file. 2. Loader diagnostic messages on the LIST file. 3. Name map on the LIST file.
Optional Output	1. Storage print on the LIST file. 2. Punched-card object program in the absolute format on the OUTPUT file.
Required User Assignments	None
Required System Control Program Control Card	LOADER RUN
Required Loader Control Card	$EXECUTION $NO EXECUTION
Loader Output Option Control Cards	$ABSOLUTE DECK [three-character file name] $STORAGE PRINT [three-character file name] $NO NAME MAP

Figure 39. Summary of a Normal LOADER RUN Job

Input	Punched-card object program in the absolute format on the CONTROL file, or object program in the absolute format on the SYSTEM file.
System Control Program Control Card	[user-comments] UPDAT three-character phase name, INSERT or [user comments] UPDAT three-character phase name, DELETE

Note: When the UPDAT INSERT card is used in a tape system, three-character phase name is the name of the phase after which the new phase is to be added.

● Figure 40. Summary of a Normal User-Update Job

Input	Object program in the absolute format on the SYSTEM file
Required User Assignments	The unit(s) referenced by the object program
Required System Control Program Control Card	PRODUCTION RUN three-character phase name

Figure 41. Summary of a Normal PRODUCTION RUN Job

ASGN Card Formats			Assumed Assignment		Remarks
Label Field (Columns 6–15)	Operation Field (Columns 16–20)	Operand Field (Columns 21–72)	Compilation (FORTRAN RUN)	Execution (LOADER RUN or PRODUCTION RUN)	
SYSTEM	ASGN	{ 1311 UNIT n / 1301 UNIT 0 / TAPE UNIT n }	1311 unit: user-assigned 1301 unit: must be assigned to UNIT 0 Tape unit: user-assigned	1311 unit: user-assigned 1301 unit: must be assigned to UNIT 0 Tape unit: user-assigned	If the system residence is 1311 or 1301, the SYSTEM ASGN card is the only required ASGN card. It must follow the Card Boot in a stack of jobs. Any other SYSTEM ASGN cards in the stack are flagged and bypassed. If the user desires that the Fortran system use less than the number of core storage positions available in the processor machine, punch a comma in column 32 and 12K or 16K beginning in column 34. If the system residence is tape and the tape unit is 1, neither the Card Boot nor the SYSTEM ASGN card is required. (Pressing the TAPE LOAD key achieves the same purpose as the Card Boot and the SYSTEM ASGN card.) If the unit is 2, 3, 4, 5, or 6, both the Card Boot and the SYSTEM ASGN card are required to start systems operations.
CONTROL	ASGN	{ READER n / CONSOLE PRINTER }	READER 1	READER 1	If the CONTROL file and the INPUT file are assigned to the card reader, the assignment must be to the same card reader.
MESSAGE	ASGN	{ PRINTER n / CONSOLE PRINTER }	PRINTER 2	PRINTER 2	When the MESSAGE file is assigned to the CONSOLE PRINTER, carriage control characters used with the 1403 or 1443 printer may appear in the message. If the MESSAGE file and the LIST file are assigned to the printer, the assignment must be to the same printer. The MESSAGE file is equivalent to Fortran file 0.
LIST	ASGN	{ PRINTER n / 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	PRINTER 2	PRINTER 2	If the LIST file and the MESSAGE file are assigned to the printer, the assignment must be to the same printer. The LIST file is equivalent to Fortran file 3.
INPUT	ASGN	{ READER n / 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	READER 1	READER 1	If the INPUT file and the CONTROL file are assigned to the reader, the assignment must be to the same reader. The INPUT file is equivalent to Fortran file 1.
OUTPUT	ASGN	{ PUNCH n / 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	OMIT	PUNCH 4 (1401 and 1460 systems) PUNCH 1 (1440 systems)	The OUTPUT file is equivalent to Fortran file 2.
LIBRARY	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 012000, END 013900 1301 UNIT 0, START 012000, END 013900 TAPE UNIT 1	1311 UNIT 0, START 012000, END 013900 1301 UNIT 0, START 012000, END 013900 TAPE UNIT 1	1311 is assumed if the SYSTEM file is assigned to 1311. 1301 is assumed if the SYSTEM file is assigned to 1301. Tape is assumed if the SYSTEM file is assigned to tape. At execution time (LOADER RUN), the LOADER file can be assigned to READER n. If the MESSAGE, LIST, and WORK5 files are assigned to a printer, the assignment must be to the same printer. The WORK1 file is equivalent to Fortran file 4. The WORK2 file is equivalent to Fortran file 5. The WORK3 file is equivalent to Fortran file 6. The WORK4 file is equivalent to Fortran file 7. The WORK5 file is equivalent to Fortran file 8. The WORK6 file is equivalent to Fortran file 9.
LOADER	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / READER n / OMIT }	1311 UNIT 0, START 010000, END 012000 1301 UNIT 0, START 010000, END 012000 TAPE UNIT 3	1311 UNIT 0, START 010000, END 012000 1301 UNIT 0, START 010000, END 012000 TAPE UNIT 3	
WORK1	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 006400, END 009600 1301 UNIT 0, START 006400, END 009600 TAPE UNIT 4	1311 UNIT 0, START 006400, END 007200 1301 UNIT 0, START 006400, END 007200 TAPE UNIT 4	
WORK2	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 006400, END 009600 1301 UNIT 0, START 006400, END 009600 TAPE UNIT 5	1311 UNIT 0, START 007200, END 008000 1301 UNIT 0, START 007200, END 008000 TAPE UNIT 5	
WORK3	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n }	1311 UNIT 0, START 006400, END 009600 1301 UNIT 0, START 006400, END 009600 TAPE UNIT 4	1311 UNIT 0, START 008000, END 008500 1301 UNIT 0, START 008000, END 008500 TAPE UNIT 6	
WORK4	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	OMIT	1311 UNIT 0, START 008500, END 009000 1301 UNIT 0, START 008500, END 009000 OMIT for tape systems	
WORK5	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / PRINTER n / OMIT }	OMIT	1311 UNIT 0, START 009000, END 009500 1301 UNIT 0, START 009000, END 009500 OMIT for tape systems	
WORK6	ASGN	{ 1311 UNIT n, START nnnnnn, END nnnnnn / 1301 UNIT n, START nnnnnn, END nnnnnn / TAPE UNIT n / OMIT }	OMIT	1311 UNIT 0, START 009500, END 010000 1301 UNIT 0, START 009500, END 010000 OMIT for tape systems	

● Figure 42. ASGN Card Formats and Assumed Assignments

Device Entry and Values of n and nnnnnn	Remarks
{1311 / 1301} UNIT n, START nnnnnn, END nnnnnn n is the number of the disk unit, and can be 0, 1, 2, 3, or 4. nnnnnn is a disk address.	The END address is the address of the next available sector. The values of nnnnnn must adhere to the following rules: 1. WORK1 and WORK2 Files. If the disk unit is 1311, the START address must be a multiple of 200. If the disk unit is 1301, the START address must be a multiple of 800. For both 1311 and 1301, the END address must be a multiple of 40. 2. WORK3, WORK4, WORK5, and WORK6 Files. For both 1311 and 1301, the START and END addresses must be multiples of 10. 3. LIBRARY File. For both 1311 and 1301, the START and END addresses must be multiples of 20. If these rules are violated, the system automatically narrows in the disk area to an area that does adhere to these rules.
TAPE UNIT n n is the number of the tape unit, and can be 1, 2, 3, 4, 5, or 6.	
READER n For 1402, n can be 0, 1, or 2. For 1442, n can be 1 or 2.	For 1402, n represents the pocket into which the cards are stacked. For 1442 or 1444, n represents the number of the unit.
PUNCH n For 1402, n can be 0, 4, or 8. For 1442, n can be 1 or 2. For 1444, n must be 3.	
PRINTER n n can be 1 or 2.	n represents the number of print positions available on the 1403 or 1443. For 1403, a 1 indicates 100 positions and a 2 indicates 132 positions. For 1443, a 1 indicates 120 positions and a 2 indicates 144 positions.
CONSOLE PRINTER	The console printer must be an IBM 1447 without a buffer feature.
OMIT	Select this option when the file is not to be used by the Fortran system. The LIST, OUTPUT, LOADER, WORK4, WORK5, and WORK6 files can be omitted.

Figure 43. Valid Device Entries

Name of Card	Label Field (Columns 6–15)	Operation Field (Columns 16–20)	Operand Field (Columns 21–72)
Copy (Tape)		COPY	[Any message and/or identification]
Halt		HALT	[Any message and/or identification]
Initialize	FORTRAN	INIT	[Any message and/or comment]
Note		NOTE	Any message and/or instruction
Pause		PAUSE	Any message and/or instruction
Run	FORTRAN	RUN	
	LOADER	RUN	
	PRODUCTION	RUN	Three-character phase name
	LIBRARY	RUN	
Update *	[User comments]	UPDAT	Three-character phase name, INSERT
	[User comments]	UPDAT	Three-character phase name, DELETE
Cards associated	LIBRARY	RUN	
with **		BUILD	nnn, where nnn specifies a name-table length that differs from 030.
LIBRARY RUN			
cards		LIST	HEADERS
***	name	LIST	
		LIST	
		INSER	
***	name	DELET	
****		COPY	
		END	

* When the UPDAT INSERT card is used in a tape system, three-character phase name is the name of the phase after which the new phase is to be added.

** This option applies only to libraries that reside on disk.

*** name is the six-character name of a subprogram.

**** This option applies only to libraries that reside on tape.

● Figure 44. System Control Cards

Type of Control Card	Format - Columns 1 - ?	Remarks
Compiler option control cards	$INTEGER SIZE = nn	01 ≤ nn ≤ 20; assumed size is 05.
	$REAL SIZE = nn	02 ≤ nn ≤ 20; assumed size is 08.
	$OBJECT MACHINE SIZE = nnnnn	11999 ≤ nnnnn ≤ 15999; assumed size is 11999.
	$MULTIPLY DIVIDE	Multiply/divide feature assumed to be present on object machine.
	$NO MULTIPLY DIVIDE	
	$PHASE NAME = name	name is 3 alphameric characters; assumed name is ///.
Compiler output option control cards	$LIST	Source program listing is assumed.
	$NO LIST	
	$NAME DICTIONARY	Name dictionary is assumed.
	$NO NAME DICTIONARY	
	$SEQUENCE NUMBER DICTIONARY	Sequence number dictionary is assumed.
	$NO SEQUENCE NUMBER DICTIONARY	
	$DICTIONARY	Both the name and sequence number dictionaries are assumed.
	$NO DICTIONARY	
Loader output option control cards *	$ABSOLUTE DECK [three-character file name]	No absolute deck is assumed. If an absolute deck is specified, OUTPUT file is assumed.
	$NO ABSOLUTE DECK	
*	$STORAGE PRINT [three-character file name]	No storage print is assumed. If a storage print is specified, LIST file is assumed. The storage print uses a print line of 120 positions.
	$NO STORAGE PRINT	
*	$NAME MAP [three-character file name]	Name map is assumed output on the LIST file.
	$NO NAME MAP	
	$INCLUDE LDR [three-character main program name or six-character subprogram name]	1 ≤ i ≤ 6 The $INCLUDE card is punched as follows. Columns / Contents 1-8 / $INCLUDE 16-18 / LDR or INP or WKi 21-23 / main program name or / or 21-26 / subprogram name
	$INCLUDE INP [three-character main program name or six-character subprogram name]	
	$INCLUDE WKi [three-character main program name or six-character subprogram name]	
	$EXECUTION	One of these cards must be supplied by the user as the last card of a LOADER RUN.
	$NO EXECUTION	

* When used, [three-character file name] appears in columns 21-23 of the Loader output option control card.

● Figure 45. Fortran Option Control Cards

Appendix II

The name, identification, and function of each phase in the Fortran system are given in the following sections.

System Control Program — Disk Resident

This section describes the phases that make up the System Control Program for disk-resident systems.

Phase Name	*ID*	*Function*
SYB	50S01	1. Determines machine size. 2. Initializes switches according to the type of reader, punch, and printer (serial or parallel). 3. Reads in the I/O package. 4. Calls the determiner.
FHW	50S11	Contains the assumed assignments for the logical files.
IOP	50S21	1. Reads or writes disk in the move or load mode. The mode depends on the processor operation. 2. Determines whether the user has exceeded specified file limits. 3. Branches to the processor phase, or branches to the end-of-file routine if the end-of-file has been sensed.
SU0	50S31	Reads in the specified phase from disk storage and branches to the specified phase.
SU1	50S41	
SU2	50S51	
SU3	50S61	
SU4	50S71	
SU5	50S81	
SU6	50S91	
OP1	50SA1	Initializes the specified area with a twenty-character control word. This control word is obtained from the temporary file-hardware table.
OP2	50SB1	
DET	50SC1	Reads the CONTROL file until a control card (HALT, PAUSE, NOTE, INIT, UPDAT, RUN, or ASGN) is sensed. When a control card is sensed, the determiner causes a halt or pauses, prints out a note, calls the update determiner, calls the selector, or calls the configurator, depending upon the type of card.
PIT	50SD1	Contains the locations of the phases in the system.
CFG	50SE1	Updates the temporary file-hardware table as specified by the ASGN card(s).
SEL	50SF1	Initializes the files used by the processor being called, and calls the first phase of that processor.
UPD	50SG1	Determines the type of update operation being performed, and calls in that particular updater.
UIN	50SH1	Places a new phase on the SYSTEM file in any available location.
UHD	50SI1	Updates the header of a phase that is in the SYSTEM file, as specified by a header card.
UDL	50SJ1	Deletes a phase from the SYSTEM file.
UPT	50SK1	Patches a part of a phase on the SYSTEM file.
DMP	50SL1	Prints storage on the LIST file.
DM2	50SM1	
F/P	50SN1	Prints all WORK files on the LIST file.
F/2	50SO1	
F/3	50SP1	
MNE	AUMNE	These phases are used by the Autocoder Assembler Program.
2XB	EX2XB	
4XB	EX4XB	
6XB	EX6XB	
8XB	EX8XB	

System Control Program — Tape Resident

This section describes the phases that make up the System Control Program for tape-resident systems.

Phase Name	*ID*	*Function*
SYB	50S01	1. Determines machine size. 2. Initializes switches according to the type of reader, punch, and printer (serial or parallel). 3. Reads in the I/O package. 4. Calls the determiner.
IOP	50S21	1. Contains the assumed assignments for the logical files. 2. Reads or writes tape in the move or load mode. The mode depends on the processor operation.

Phase Name	*ID*	*Function*
		3. Branches to the processor phase, or branches to the end-of-file routine if the end-of-file has been sensed. 4. Reads in the specified phase from the system tape and branches to the specified phase. 5. Initializes the specified area with a twenty-character control word. This control word is obtained from assumed assignments for logical files.
DET	50SC1	1. Reads the CONTROL file until a control card (HALT, PAUSE, COPY, NOTE, INIT, UPDAT, RUN, or ASGN) is sensed. When a control card is sensed, the determiner causes a halt, or pauses, or prints out a note. 2. Updates assumed file assignments as specified by the ASGN card(s). 3. Initializes the files used by the processor being called, and calls the first phase of that processor. 4. Determines the type of update operation being performed, and calls the updater.
UPD	50SH1	1. Places a new phase on the SYSTEM file in any available location. 2. Updates the header of a phase that is in the SYSTEM file, as specified by a header card. 3. Deletes a phase from the SYSTEM file. 4. Patches a part of a phase on the SYSTEM file.
DMP DM2	50SL1 50SM1	Prints storage on the LIST file.
F/P	50SN1	Prints all WORK files on the LIST file.

Fortran Processor Program

This section describes the phases that make up the Fortran Processor Program.

Phase Name	*ID*	*Function*
00F	00FIV	1. Reads compiler option control cards, if any, from the INPUT file and outputs them on the LIST file. 2. Initializes deblocking routines (GETEX and PUTEX).
		3. Sets up an area in upper core storage for GETEX and PUTEX buffers. 4. Passes the first source-program statement to phase 10F.
10F	10FIV	1. Reads source statements from INPUT file until the END card is sensed. 2. Assigns sequence numbers to all source statements except comments statements. 3. Outputs the source program listing on the LIST file, if the option is exercised. 4. Replaces key words (COMMON, DIMENSION, GO TO, etc.) with internal three-character symbols. Replaces remaining portion of each statement with internal symbols. 5. Replaces unrecognizable statements with diagnostic codes. 6. Outputs nonexecutable and I/O name lists on WORK2 and executable statements on WORK1. 7. Calls phase 20F, 25F, or 30F, depending on type of source statements.
20F	20FIV	1. Extracts DIMENSION, COMMON, EQUIVALENCE, FUNCTION, SUBROUTINE, EXTERNAL, and type statements from WORK2. 2. Builds a name-attribute table in upper core storage.
21F	21FIV	1. Uses name-attribute table built by phase 20F and allocates object-time storage for COMMON variables and applicable EQUIVALENCE definitions. 2. Allocates storage for normal variables with EQUIVALENCE or DIMENSION definitions, and adds this information to the name-attribute table. No storage is allocated unless the complete set of variable attributes has been determined. 3. Compresses the name-attribute table, deleting information no longer required. 4. Outputs a macro reflecting ordering or various variable types within COMMON on the PUTEX file. 5. Calls phase 25F, if required. Otherwise, phase 30F is called.

Phase Name	*ID*	*Function*
25F	25FIV	1. Extracts FORMAT, DATA, and DEFINE FILE statements and I/O name lists from WORK2. 2. Replaces names in DATA or I/O name lists and source or generated constants from I/O name lists with object-time addresses. 3. Generates sequence of macros with associated parameters on the PUTEX file for each statement. An increased object-time character count is included with each sequence number or label assignment macro reflecting the total number of object-time characters since the last sequence number or label assignment macro. 4. Begins label table for FORMAT statement numbers. 5. Continues building name-attribute table and continues allocating storage as required.
30F	30FIV	1. Extracts executable statements from the GETEX file, which can contain macros from phase 25F. 2. Continues building name-attribute table and label table, with allocation accomplished when required. An address is substituted for a name and an internal representation is substituted for a label. 3. Generates macros with appropriate parameters for all executable statements, except for expressions that are passed to phase 40F. Subscripted variables are processed as in phase 25F, generating macros even when they are part of an expression. The object-time incremental character count is included with sequence number or label assignment macros. 4. If source label and name table overflows, subsequent source labels and/or names are passed in expanded form for replacement by phase 34F.
34F	34FIV	Phase 34F is an optional phase. When required, it completes the storage allocation for normal variables. It replaces names with addresses, and source labels with internal label notation.
35F	35FIV	1. Outputs a name dictionary on the LIST file, if requested. 2. Outputs constants as macros in the name table on the PUTEX file.
36F	36FIV	When a subprogram is being compiled, macros are generated on the PUTEX file to represent the necessary processing when the subprogram is called. A "prologue" is generated; also, an "epilogue" is generated which represents any necessary resetting of values before returning control to the calling program. If variable dimensions have been used, a pass on the GETEX file is required when building the prologue name to indicate additional object-time calculations.
40F	40FIV	1. Extracts expressions from the GETEX file that were partially processed by phase 30F. The expressions are reordered according to implied operator precedence and parentheses, and macros are generated on the PUTEX file. 2. Extracts label assignments and sequence number macros from the GETEX file. For a sequence number, the incremental character count is replaced with an actual accumulative character count, and the macro is passed. Label assignment macros are not passed further. Actual addresses for source labels and generated labels are entered into the label table and generated label table. 3. Passes control to phase 45F if a DATA statement is sensed.
45F	45FIV	1. Extracts DATA name list macro sequences from the GETEX file. The macros are expanded into object code in a phase work area. The code references some included subroutines. DATA literal list macros are matched with object-time addresses and passed as regular literal macros. 2. Continues label assignments and sequence number processing as in phase 40F.
53F	53FIV	1. This phase is called only if diagnostic codes were output by a previous phase. The codes are extracted and translated into diagnostic

Phase Name	*ID*	*Function*
		messages that are output on the LIST file. These errors can be warnings, or severe errors that would prevent a successful compilation. 2. If the messages were merely warnings, phase 70F is called. 3. If the messages indicated errors that would prevent a successful compilation, the system halts, then control reverts to the DET (determiner) phase of the System Control Program, which will read a control card from the CONTROL file.
70F	70FIV	Generates object code in the relocatable format from internal macros and associated parameters. Addresses are substituted for label references. A header card image is always generated first, and a trailer card image is always generated as the last card. Between these two card images will be relocatable and external name cards to indicate object characters to be loaded and interprogram references. Card images appear on the LOADER and/or OUTPUT files, depending on actual device assignments. Control is returned to the DET (determiner) phase of the System Control Program.
78F	78FIV	Builds, updates, or lists the Fortran relocatable subprogram library.
79F	79FIV	1. Reads loader control cards from the CONTROL file. 2. Relocates and loads a main program and required subprograms, possibly from the subprogram library. 3. Establishes interprogram communication by replacing external references by actual addresses.
80F	80FIV	Produces an external name map on the LIST file, if requested.
81F	81FIV	Produces storage print of the loaded program on the LIST file, if requested. The storage print does not include the standard overlay package.
82F	82FIV	Produces an absolute deck on the OUTPUT file, if requested.
90F	90FIV	Standard overlay package for 1401 or 1460, including arithmetic interpreter and standard I/O routines.
91F	91FIV	Standard overlay package for 1440, including arithmetic interpreter and standard I/O routines.

Appendix III

Building a System that Contains Fortran and Autocoder

In this section, the Autocoder system refers to *1401/1440/1460 Autocoder (on Disk),* program number 1401-AU-008. The specifications and operating procedures for this program are contained in the Systems Reference Library publications, *Autocoder (on Disk) Language Specifications for IBM 1401, 1440, and 1460,* Form C24-3258 and *Autocoder (on Disk) Program Specifications and Operating Procedures for IBM 1401, 1440, and 1460,* Form C24-3259.

File Considerations

Because the System Control Program is the controlling element of the Autocoder system as well as the Fortran system, it is possible to build a SYSTEM file that contains both the Fortran Processor Program and the Autocoder Assembler Program.

Figure 46 shows the disk-storage allocation on the system unit when both Fortran and Autocoder are present.

The user should consult the referenced Autocoder publications for a description of the Autocoder system.

Two differences exist when the Autocoder system resides alone on a disk unit as opposed to when it resides on a disk unit that also contains the Fortran system. These differences are:

1. The assumed assignment of the Autocoder LIBRARY file.
2. The assumed assignments of the Autocoder WORK files.

Figure 47 gives the assumed assignments of the Autocoder LIBRARY file and the Autocoder WORK files when Autocoder and Fortran reside on the same SYSTEM file. Do *not* consider the assumed assignments for these files as given in the Autocoder operating procedures publication.

Note also that Autocoder uses a maximum of three WORK files, whereas six WORK files are defined for Fortran. Further, the user should note that Autocoder defines a CORELOAD file (not applicable to Fortran), and Fortran defines a LOADER file (not applicable to Autocoder). The assumed assignments for the remaining logical files (CONTROL, MESSAGE, LIST, INPUT, and OUTPUT) are the same for both the Autocoder and the Fortran systems.

File	Mode	File-Protected	Sector Range
SYSTEM File			
Autocoder Preprocessor Work Area	Move	No	000000-000089
Autocoder Preprocessor	Move	No	000090-000199
Autocoder Preprocessor	Load	No	000200-000259
Autocoder Preprocessor	Move	No	000260-000299
Autocoder Preprocessor	Load	No	000300-000899
Not Used	Load	No	000900-002499
System Control Program	Load	Yes	002500-002904
Fortran Processor Program	Load	Yes	002905-002979
Not Used	Load	Yes	002980-002999
System Control Program	Load	Yes	003000-003175
Fortran Processor Program	Load	Yes	003176-004200
Autocoder Assembler Program	Load	Yes	004201-005529
Area for User's Fortran Object Program Library	Load	Yes	005530-006399
WORK Files	Move	No	006400-009999
LOADER File (Fortran only. This area is used by Autocoder as a continuation of the WORK files.)	Move	No	010000-011999
LIBRARY File (Fortran)	Move	No	012000-013899
LIBRARY File (Autocoder)	Move	No	013900-019979

● Figure 46. Disk Storage Allocation